UNIVERSITY OF ALBERTA

A UNIFIED MODELING METHODOLOGY FOR SIMULATION-BASED PLANNING OF CONSTRUCTION PROJECTS

BY

DANY HAJJAR ©

A thesis submitted to the Faculty of Graduate Studies in partial fulfillment of the

requirements for the degree of **DOCTOR OF PHILOSOPHY**

in

Construction Engineering and Management

DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING

EDMONTON, ALBERTA

Fall 1999

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-46845-3

Canada

UNIVERSITY OF ALBERTA

Library Release Form

Name of Author: Dany Hajjar

Title of Thesis: A Unified Modeling Methodology For Simulation-Based Planning of Construction Projects

Degree: Doctor of Philosophy

Year this Degree Granted: 1999

Permission is hereby granted to the University of Alberta to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly, or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

13332-89 St
Edmonton, Alberta
Canada T5E-3K2

Date Submitted: June 29, 1999

UNIVERSITY OF ALBERTA

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled "**A Unified Modeling Methodology For Simulation-Based Planning of Construction Projects**" by **Dany Hajjar** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in Construction Engineering and Management.

Dr. S. AbouRizk

Dr. S. Ariaratnam

Dr. H. Hoover

Dr. T. Hrudey

Dr. A. Oloufa

DATE: 6/28/99

This thesis is dedicated with love, admiration, and respect to my parents

Nabil and Simone Hajjar

I am but a product of your dreams and sacrifices

ABSTRACT

Computer simulation is a proven technology and countless studies have been performed to demonstrate its applicability to the analysis of a wide range of construction operations including aggregate production, earth-moving, mining, and tunneling. The presented research describes a methodology that leads to improvements in the appeal of simulation-based tools within the construction industry and the simplification of their development process.

A study was first undertaken to investigate and develop various techniques, concepts and methods of potential benefit as related to the research objective. This study resulted in the successful development and implementation of three custom simulation tools for earth-moving, aggregate production and site dewatering operations. An analysis of the success factors of these tools as well as their limitations led to the identification of a critical set of features. This set was used as the basis for the formalization of five concepts: special purpose simulation modeling, graphical modeling, integrated modeling, modular and hierarchical modeling, and the hybrid tool development and utilization approach. This set of concepts was then combined, with the help of object-oriented modeling concepts, into a unified modeling methodology.

A computer system called Simphony was then developed based on the unified modeling methodology. The design and implementation of Simphony was guided by the principles of object-oriented application frameworks. Case study results showed that novice developers are able to produce new Simphony-based tools at a pace that exceeds that of a programmer using a commercial development system by a factor of six. Further results from other studies showed that the simplification factor can be as high as sixteen.

ACKNOWLEDGEMENTS

I would like to start by thanking all the industry members who have collaborated on the various research projects related to my thesis. This includes Lafarge Canada, North American Construction, and PCL. The financial and technical support of these companies is greatly appreciated.

Many great thanks to all the graduate CEM students that I have ever known for all their support. Special thanks to Dr. Nader Chehayeb, Jianfei Xu and the entire 1999 Civil Engineering 606 class for their help with the testing of Simphony.

My thanks to all the professors that have contributed to my education over the years; in particular I am grateful to Professor Peter Dozzi, Professor William Weir, Dr. Amina Fayek, and Dr. Sam Ariaratnam.

The suggestions and thoughtful reviews of my dissertation by my committee members, Sam Ariaratnam, Terry Hrudey, James Hoover, and Amr Oloufa are greatly appreciated.

My most sincere thanks and gratitude to my supervisor, friend, and mentor, Dr Simaan AbouRizk, who took me under his wing, treated me as his son, and showed endless patience and encouragement towards me throughout the whole dissertation process. His advice and confidence were essential for the completion of this thesis. Certainly, this dissertation could not have been started, let alone finished, without his input, guidance, trust, and endless scientific and moral wisdom. There is no part of this work that has not been the focus of his penetrating mind, and there has been no question posed to him to which he has failed to apply his full intellect and produce a suitable answer. At first, I didn't agree with all the answers. But it did not take long to realize that he was always right. Dr. Simaan AbouRizk, I thank you from the bottom of my heart.

Funding for my research was partly provided by the University of Alberta, Government of Alberta, Natural Science and Engineering Research Council of Canada, and the National Research Council.

Table of Contents

CHAPTER 1 – INTRODUCTION 1

1.1 INTRODUCTION 1
1.2 COMPUTER SIMULATION AS A PROJECT PLANNING TECHNIQUE 2
1.3 RESEARCH OBJECTIVES 2
1.4 RESEARCH SUMMARY 3
1.5 THESIS ORGANIZATION 4

CHAPTER 2 – BACKGROUND 5

2.1 PROJECT PLANNING TECHNIQUES 5
2.2 GENERAL COMPUTER SIMULATION 7
2.3 CONSTRUCTION SIMULATION 12

CHAPTER 3 – EARTH-MOVING PROJECT SIMULATION 15

3.1 INTRODUCTION 15
3.2 BACKGROUND 15
3.3 THE OVERALL SYSTEM STRUCTURE 18
3.4 DESIGN AND DEVELOPMENT OF AP2-EARTH 19
3.5 CASE STUDY 34
3.6 CONCLUSION 42

CHAPTER 4 – AGGREGATE PRODUCTION PLANT SIMULATION 43

4.1 INTRODUCTION 43
4.2 AGGREGATE PRODUCTION 43
4.3 SIMULATION OF AGGREGATE PROCESSING 49
4.4 MODEL VALIDATION 65
4.5 CONCLUSION 69

CHAPTER 5 – CONSTRUCTION SITE DEWATERING SIMULATION 71

5.1 INTRODUCTION 71
5.2 BACKGROUND 71
5.3 SYSTEM DESCRIPTION 73
5.4 CASE STUDY 86
5.5 CONCLUSIONS 92

CHAPTER 6 – PHASE ONE RESEARCH FINDINGS 94

6.1 INTRODUCTION 94
6.2 USER INTERFACE 94
6.3 MODELING PHILOSOPHY 95
6.4 SIMULATION EXECUTION APPROACH 96
6.5 SIMULATION RESULT REPRESENTATION 98
6.6 INTEGRATION REQUIREMENTS 99
6.7 CONCLUSIONS 99

CHAPTER 7 – UNIFIED MODELING METHODOLOGY 101

7.1 INTRODUCTION 101
7.2 SPECIAL PURPOSE SIMULATION MODELING 102
7.3 GRAPHICAL MODELING 107
7.4 INTEGRATED MODELING 110
7.5 MODULAR AND HIERARCHICAL MODELING 115
7.6 ISSUES OF TOOL DEVELOPMENT AND UTILIZATION 122
7.7 OBJECT-ORIENTED MODELING 125
7.8 UNIFIED MODELING METHODOLOGY 126
7.9 CONCLUSIONS 130

CHAPTER 8 – SIMPHONY ENVIRONMENT 131

8.1 INTRODUCTION 131
8.2 SPS TEMPLATE DEVELOPMENT AND THE SIMPHONY DESIGNER 132
8.3 CONSTRUCTION SIMULATION USING THE SIMPHONY EDITOR 182
8.4 SUMMARY 185

CHAPTER 9 – SIMPHONY APPLICATION FRAMEWORK 187

9.1 BACKGROUND 187
9.2 INTRODUCTION 187
9.3 OVERVIEW OF APPLICATION FRAMEWORK THEORY 189
9.4 APPLICATION FRAMEWORKS FOR CONSTRUCTION SIMULATION 191
9.5 CONCLUSIONS 201

CHAPTER 10 – CASE STUDIES 202

10.1 INTRODUCTION 202
10.2 AP2-EARTH REDEVELOPMENT 203
10.3 CRUISER REDEVELOPMENT 209
10.4 CLASS EXPERIMENTS 216
10.5 SUMMARY 219

CHAPTER 11 – FINAL DISCUSSION 220

11.1 RESEARCH SUMMARY 220
11.2 SUMMARY OF RESEARCH CONTRIBUTIONS 223
11.3 RECOMMENDATION FOR FUTURE DEVELOPMENT 224

BIBLIOGRAPHY 225

APPENDIX 1 – DEVELOPMENT CODE FOR THE COMMON TEMPLATE 233

APPENDIX 2 – SIMPHONY BUILD HISTORY 253

APPENDIX 3 – CEM_EMS TEMPLATE CODE 255

List of Tables

TABLE 2-1 SAMPLE DISCRETE-EVENT SIMULATION PROCESSING SESSION 9

TABLE 3-1 DESCRIPTION OF AP2-EARTH CLASSES 21

TABLE 3-2 EVENT FLOW DIAGRAMMING LEGEND 28

TABLE 3-3 OBSERVED IDEAL PRODUCTION OF EXCAVATORS 36

TABLE 3-4 TRUCK CHARACTERISTICS 36

TABLE 3-5 COMPARISON OF ACTUAL AND PREDICTED TRAVEL DURATIONS FOR SCENARIO 1 39

TABLE 3-6 COMPARISON OF ACTUAL AND PREDICTED WAITING DURATION FOR SCENARIO 1 40

TABLE 3-7 COMPARISON OF ACTUAL AND PREDICTED SYSTEM PRODUCTIONS FOR SCENARIO 1 40

TABLE 3-8 COMPARISON OF ACTUAL AND PREDICTED TRAVEL DURATIONS FOR SCENARIO 2 41

TABLE 3-9 COMPARISON OF ACTUAL AND PREDICTED WAITING DURATION FOR SCENARIO 2 41

TABLE 3-10 COMPARISON OF ACTUAL AND PREDICTED SYSTEM PRODUCTIONS FOR SCENARIO 2 41

TABLE 4-1 CRUISER ABSTRACT CLASSES 50

TABLE 4-2 CONTENTS OF EVENT QUEUE 63

TABLE 4-3 RAW FEED GRADATION 66

TABLE 8-1 MODELING ELEMENT BEHAVIORS AND ASSOCIATED PROPERTIES, METHODS AND EVENTS 134

TABLE 8-2 LIST OF SIMPHONY SERVICES, THEIR ACCESS POINTS, PROPERTIES AND METHODS 150

TABLE 8-3 GUI CONTROLS USED FOR THE EXTERNAL REPRESENTATION OF ELEMENT ATTRIBUTES 159

TABLE 8-4 SIMPHONY TEMPLATE DESIGN GUIDELINES 167

TABLE 8-5 GUIDELINES FOR THE INCREMENTAL DEVELOPMENT OF A MODELING ELEMENT 168

TABLE 8-6 LIST OF DEVELOPED MODELING ELEMENTS USED TO SUPPORT GPS-BASED MODELING 179

TABLE 9-1 COMMON FEATURES ACROSS CONSTRUCTION SIMULATION TOOLS 194

TABLE 9-2 LIST OF FACTORED FEATURES ACROSS TOOLS 194

TABLE 9-3 LIST OF VARIABLE FEATURES ACROSS TOOLS 195

TABLE 9-4 SAMPLE ATTRIBUTE TABLE FOR TRUCK MODELING ELEMENT 198

TABLE 10-1 FUNCTION OF EACH CEM_EMS TEMPLATE ELEMENT 205

TABLE 10-2 DEVELOPMENT HOURS FOR THE CEM_EMS TEMPLATE COMPONENTS 208

TABLE 10-3 DEVELOPMENT HOURS FOR THE CEM_CRUSH TEMPLATE COMPONENTS 216

TABLE 10-4 SUMMARY OF STUDENTS' CYCLONE TEMPLATE DEVELOPMENT 217

TABLE 10-5 SUMMARY OF STUDENTS' PROJECT TEMPLATE DEVELOPMENTS 219

List of Figures

FIGURE 2-1 SAMPLE DISCRETE-EVENT MODEL REPRESENTATION 8

FIGURE 3-1 AP2-EARTH MODULES 18

FIGURE 3-2 AP2-EARTH CLASS HIERARCHY 20

FIGURE 3-3 CINTERSECTION GRAPHICAL BEHAVIOR 22

FIGURE 3-4 CTRUCK GRAPHICAL BEHAVIOR 23

FIGURE 3-5 CPROJECT GRAPHICAL BEHAVIOR 23

FIGURE 3-6 CTRUCK DATA ANALYSIS BEHAVIOR 25

FIGURE 3-7 CTRUCK REPORTING BEHAVIOR 25

FIGURE 3-8 SAMPLE TRAVEL TIME CURVE 27

FIGURE 3-9 EXAMPLE INTERSECTION CROSSING SCENARIOS 27

FIGURE 3-10 HAULING SIMULATION MODEL 29

FIGURE 3-11 EXAMPLE EARTH-MOVING PROJECT 30

FIGURE 3-12 CPROJECT SIMULATION MODEL 31

FIGURE 3-13 INTEGRATION WITH ESTIMATE MODULE 34

FIGURE 3-14 OVERBURDEN CLEARING PROJECT SITE LAYOUT 35

FIGURE 3-15 OBSERVED DUMPING TIME 37

FIGURE 3-16 TRAFFIC FLOW AROUND LOADING AREA 38

FIGURE 3-17 AP2-EARTH MODEL REPRESENTATION 39

FIGURE 4-1 TYPICAL AGGREGATE PLANT LAYOUT 46

FIGURE 4-2 EXAMPLE OF CLOSED SYSTEM OPERATIONS 47

FIGURE 4-3 SAMPLE SIEVE ANALYSIS RESULTS 48

FIGURE 4-4 CRUISER CLASS HIERARCHY 50

FIGURE 4-5 SCREEN DIALOG BOX 51

FIGURE 4-6 CONVEYER DIALOG BOX 52

FIGURE 4-7 MAIN MODELING COMPONENTS 53

FIGURE 4-8 SAMPLE CRUSHER PERFORMANCE CHART 56

FIGURE 4-9 ALLIS-CHALMER RELATIONSHIP FOR CALCULATING SCREENING EFFICIENCY 58
FIGURE 4-10 DISTRIBUTION OF UNDERSIZE PARTICLES BASED ON LOADING RATIO 59
FIGURE 4-11 MULTI-DECK MULTI-SPLIT SCREEN ANALYSIS PROCEDURE 60
FIGURE 4-12 EXAMPLE PLANT FLOW SIMULATION 62
FIGURE 4-13 PLANT DESIGN STEPS 64
FIGURE 4-14 RAW INPUT STREAM MODELING 65
FIGURE 4-15 PLANT LAYOUT FOR CASE STUDY 66
FIGURE 4-16 COMPARISON OF CRUISER PREDICTED OUTPUT VS ACTUAL GRADATIONS 68
FIGURE 4-17 CRUISER CASE STUDY RESULTS 69
FIGURE 5-1 CSD SYSTEM MODULES 73
FIGURE 5-2 TYPICAL AQUIFER TYPES: A) CONFINED; AND B) UNCONFINED 74
FIGURE 5-3 MAIN OBJECT DEFINITION AND MANIPULATION VIEW 78
FIGURE 5-4 CPUMPINGWELL CLASS DIALOG BOX 79
FIGURE 5-5 CBLOCK CLASS DIALOG BOX 79
FIGURE 5-6 CSITE CLASS DIALOG BOX 80
FIGURE 5-7 CSITE CROSS-SECTIONAL VIEW 81
FIGURE 5-8 SOIL PROFILE IN EXCAVATION AREA FOR CASE STUDY 87
FIGURE 5-9 PUMPING WELL LAYOUT FOR CASE STUDY 88
FIGURE 5-10 REPRESENTATION OF CASE STUDY DATA IN CSD 89
FIGURE 5-11 CONTOUR-VIEW OF WATER TABLE LEVEL THROUGHOUT SITE FOR CASE STUDY 91
FIGURE 6-1 TRANSLATION OF HIGH LEVEL MODEL TO DISCRETE EVENT REPRESENTATION 97
FIGURE 7-1 UNIFIED MODELING METHODOLOGY AND CONTRIBUTING CONCEPTS 101
FIGURE 7-2 SAMPLE CYCLONE SIMULATION MODEL 102
FIGURE 7-3 AN EXAMPLE OF GENERATED SIMULATION OUTPUT IN A GRAPHICAL FORMAT 106
FIGURE 7-4 EXAMPLES OF GRAPHICAL REPRESENTATION OF RELATIONSHIPS IN TWO SPS TOOLS 109
FIGURE 7-5 RELATIONAL REPRESENTATION OF GENERATED SIMULATION PLANNING DATA 115
FIGURE 7-6 USE OF CONNECTION POINTS FOR EXPOSING ENCAPSULATED SIMULATION CODE 116
FIGURE 7-7 UTILIZATION OF INVISIBLE RELATIONSHIPS FOR ROUTING ENTITIES 118

FIGURE 7-8 USE OF MODULARITY AND HIERARCHY CONCEPTS FOR LINKING MULTIPLE SPS TOOLS 120

FIGURE 7-9 EXTENDING A SPS TOOL WITH THE HELP OF GENERIC MODELING ELEMENTS 121

FIGURE 7-10 ACCOMMODATION OF USERS AND DEVELOPERS WITH VARYING DEGREES OF SKILL 123

FIGURE 7-11 BEHAVIORS OF THE GENERIC BASE MODELING ELEMENT 127

FIGURE 7-12 LIST OF IDENTIFIED SERVICES AND RELATIONSHIPS TO ELEMENT BEHAVIORS 128

FIGURE 7-13 STRUCTURE OF SIMULATION MODELS BASED ON THE UNIFIED MODELING METHODOLOGY. 129

FIGURE 8-1 SIMPHONY ENVIRONMENT 131

FIGURE 8-2 TRIGGERED EVENTS IN RESPONSE TO USER MANIPULATION OF RELATIONSHIPS 137

FIGURE 8-3 EVENTS TRIGGERED IN RESPONSE TO THE ADDITION AND DELETION OF ELEMENTS 138

FIGURE 8-4 UTILIZATION OF FILES TO REPRESENT A TRUCK QUEUING SITUATION 140

FIGURE 8-5 TRIGGERED EVENT SEQUENCE FOR SIMULATION BEHAVIOR 142

FIGURE 8-6 USE OF GEOMETRICAL ATTRIBUTES AS MODELING ELEMENT REFERENCE POINTS 147

FIGURE 8-7 SIMULATION SERVICE COMPONENTS AND ITS ACCESS POINTS 151

FIGURE 8-8 SIMULATION EVENT SCHEDULING AND PROCESSING 153

FIGURE 8-9 ATTRIBUTE MANIPULATION THROUGH THE ELEMENT ATTRIBUTE DIALOG BOX 158

FIGURE 8-10 GUI PRESENTATION OF SUMMARY STATISTICAL ANALYSIS RESULTS 160

FIGURE 8-11 SAMPLE GENERATED HISTOGRAM 160

FIGURE 8-12 SAMPLE GENERATED TIME GRAPH 161

FIGURE 8-13 TRACE NAVIGATION FORM 162

FIGURE 8-14 PROJECT PLAN SUMMARY FORM 163

FIGURE 8-15 MAIN FORM OF SIMPHONY DESIGNER 165

FIGURE 8-16 BITMAP DATABASE MANIPULATION FORM 166

FIGURE 8-17 UTILIZATION OF THE COMMON TEMPLATE TO DEFINE SIMULATION BEHAVIOR 181

FIGURE 8-18 SIMPHONY EDITOR MAIN FORM 182

FIGURE 8-19 EXPRESSION EDITOR USED IN THE CREATION OF A DYNAMIC ROAD SEGMENT 184

FIGURE 8-20 USING THE SCRIPTING CAPABILITIES TO CREATE A LARGE NUMBER OF ELEMENTS 185

FIGURE 9-1 APPLICATION FRAMEWORK APPROACH TO TOOL DEVELOPMENT 188

FIGURE 9-2 UTILIZATION OF SIMPHONY APPLICATION FRAMEWORK FOR GENERATING SPS TEMPLATES.. 191

FIGURE 9-3 SIMPLIFIED REPRESENTATION OF SIMPHONY APPLICATION FRAMEWORK 197

FIGURE 10-1 SAMPLE MODEL LAYOUT BASED ON THE CEM_EMS TEMPLATE 203

FIGURE 10-2 MODELING ELEMENTS OF THE CEM_EMS TEMPLATE 205

FIGURE 10-3 DEFINITION OF SIMULATION BEHAVIOR THROUGH GPS TEMPLATE 207

FIGURE 10-4 SAMPLE MODEL LAYOUT BASED ON THE CEM_CRUSH TEMPLATE 209

FIGURE 10-5 CEM_CRUSH TEMPLATE STRUCTURE 210

FIGURE 10-6 MODELING SIZE SEPARATION PROCESSES USING SCREEN SUB-MODELS 212

FIGURE 10-7 SAMPLE MODEL LAYOUT BASED ON STUDENT B'S PAVING TEMPLATE 218

FIGURE 10-8 SAMPLE MODEL LAYOUT BASED ON STUDENT E'S TUNNELING TEMPLATE 219

Chapter 1 – Introduction

1.1 Introduction

Construction engineering and management is a discipline that deals with the production aspects of realizing a facility from conception to delivery. Elements of construction management include:

1. Feasibility studies and economic analysis
2. Budget and cash-flow planning
3. Construction contract preparation
4. Cost and schedule estimation
5. Methods planning and analysis
6. Production analysis
7. Cost and schedule monitoring and control
8. Revenue and payment management
9. Equipment and material management

Construction projects are unique due to their complexity, scale and cost. Unlike manufacturing, every construction "product" or project is unique with regards to such things as:

1. Materials or material combinations used
2. Equipment and supplies required
3. Engineering design and requirements
4. Construction methods involved.

As a result, computer-based methods have been developed to assist construction engineers. The most common systems are those used for estimating the cost and schedule of a project. This research deals with simulation based techniques that improve the overall process of project planning, estimating and analysis.

1.2 Computer Simulation as a Project Planning Technique

The most important factor in planning a construction project is a clear and accurate understanding of the construction methods involved. During the estimating stage, this understanding translates into the predicted duration and resource requirements for the involved activities. Formal representation of this knowledge in the form of a computer model makes it possible for the planner[1] to easily experiment with different possible arrangements of the construction methods. Variations can be applied to crew compositions, activity sequences and resource types. By experimenting with different scenarios, the planner can optimize the construction operation and generate representative project plans.

1.3 Research Objectives

Despite its obvious potential, the use of computer simulation for planning construction projects has been limited mainly to academia and a few large contractors who can afford to employ dedicated simulation professionals.

---

[1] The term "Plan" or "Project Plan" is used generically to refer to all related aspects of a project plan including estimates and schedules. "Planner" is used to refer to the person responsible for preparing a project plan.

The overall objective of this research is to develop an approach that improves the appeal of computer simulation and transforms it into an accepted tool that fits naturally within the industry's information technology framework. To achieve this objective, the following sub-objectives and steps were identified:

1. Develop, test and validate three simulation-based tools for the modeling and analysis of three diverse construction methods and deploy them in an industry setting
2. Analyze the results of industry implementation and identify a set of key features such simulation tools should posses
3. Formalize the set of simulation concepts that described the identified features
4. Combine the formalized concepts into a unified modeling methodology
5. Develop, test and validate a simulation system based on the unified methodology

1.4 Research Summary

To achieve the stated objectives, research was done in two phase During the first phase, three developments were undertaken to investigate and develop various techniques, concepts and methods of potential benefit as related to the research objective. Three construction methods were chosen: earth-moving, aggregate production and site dewatering. In each case, the process fundamentals were first studied, followed by an identification of the requirements for the custom simulation tool to be developed. Based on these requirements, a computer based simulation tool was developed, validated and implemented with a local construction company. The three case studies were then analyzed together to obtain the set of features that were critical to the tools' success. The analysis also identified the limitations of the developed tools and the set of further requirements.

During the second phase of research, the collective experience gained from phase 1 and the identified set of requirements were used in the development of a unified modeling methodology. A computer application based on the developed methodology was then developed and tested.

1.5 Thesis Organization

Chapter 2 will first provide a summary of the state-of-the-art in project management techniques including computer simulation. Chapters 3 through 5 will discuss the three initial case studies done as part of the first phase of research. Chapter 6 will present the result of the case study analysis and the list of identified requirements. Chapter 7 will introduce the unified modeling methodology and its contributing concepts. Chapter 8 will present the details of the computer system based on the developed methodology. Chapter 9 will introduce an enabling technology from the software engineering discipline that was critical to the successful implementation of the software system. Chapter 10 will discuss the various tests and case studies performed to examine the flexibility, effectiveness and usability of the developed methodology. In Chapter 11, the final discussion is provided.

Chapter 2 – Background

2.1 Project Planning Techniques

There is a host of techniques for planning construction projects. Most are general project management techniques which are not specific to the construction industry. Others have been developed to deal with specific situations such as repetitive construction activities.

The basic process of planning a project involves three steps. First, activities required to complete the work are identified and their duration determined. Second, the activities are sequenced in a logical manner. Third, a project schedule is prepared.

Bar charts are the simplest form of communicating the schedule requirements of a given project. With bar charts, the most basic project activities are represented in the form of horizontal strips on a time-scaled graph. Resource requirements, relationships to other activities, and progress are sometimes shown on these charts.

Computer systems such as Primavera and Microsoft Project simplify the task of constructing and managing bar chart based plans. These systems also provide both facilities for representing large projects hierarchically and methods for leveling resources in order to calculate the total project duration.

The critical path method (CPM) is the underlying method used by these systems to calculate project duration. One of the limitations of CPM is its assumption that activity durations are constant. In reality, construction activities are affected by many factors and their duration cannot be determined with certainty. Program Evaluation and Review Technique (PERT) was developed to model this uncertainty. PERT networks are similar to CPM networks with the difference being that activity duration is represented with Beta distributions. Based on the central limit theorem, this activity information is combined to predict a normally distributed total project duration.

Monte-Carlo simulation is a more general, probabilistic approach to scheduling. Activity duration can be represented using any stochastic distribution. The total project duration is obtained by performing a number of simulation runs. On each run, the activity durations are sampled randomly and the total project duration is calculated. The result is a computer-generated distribution of the predicted project duration. This method allows for the formal representation of the element of risk in project plans.

It is worth mentioning that several specialized techniques were developed to deal with special types of construction projects, mainly those that include repetitive or cyclic activities such as the building of a high-rise or the construction of a highway. An example of such a technique is the Linear Scheduling Method (Johnston 1981).

The majority of the described methods work best when the activity resource requirements are available when needed. Some developed methods of resource allocation can be used to "level" a given schedule in order to obtain a revised schedule (Russel and Dubey 1995). However, it has been shown that these methods fail to represent certain basic realities of a construction project, particularly the way in which construction projects are characterized by complexity, dynamic interaction between resources and processes (Paulson et. al. 1987), and the varying construction methods utilized based on specific project conditions.

This led to the development of simulation based planning techniques where the construction process was represented using a formal model that allows for the analysis of dynamic situations.

2.2 General Computer Simulation

2.2.1 Introduction

Computer simulation is defined as the process of designing a mathematical-logical model of a real world system and experimenting with the model on a computer (Pristker 1986). Early simulation users were required to build a model by writing programming code, mainly in FORTRAN, and experimenting by directly manipulating the computer program. This was followed by the invention of simulation specific programming environments where users write simulation specific code or access a provided function library. "Modeling" is the term used to describe the process of specifying a given simulation model. In the next phase of development, a host of systems that allowed for alternative model development were introduced. This meant that modelers no longer had to write code directly. Graphical modeling made it possible to define the simulation model by creating, manipulating and linking a number of available basic building blocks. This meant that users no longer had to be proficient in programming. A detailed account of the history of simulation concepts and systems is found in Kreutzer (1986).

There are many ways of modeling a given problem. These generally fall into two categories; continuous and discrete-event. With continuous modeling, differential equations are used to describe the progress of a given activity. However, when mathematical modeling is not possible, the discrete event approach is utilized.

Discrete-event simulation views a model as a set of events and transitions. Transformations are processed as part of event handlers and no relevant transformations are assumed to occur during transitions. Entities represent the active elements of the model. They travel throughout the event network and trigger transformations. An important component of a discrete-event simulation is the simulation event monitor, which is responsible for managing the event calendar. The described concept is now illustrated below with a simple example. Assume that trucks arrive at a concrete batch plant and await their turn for loading fresh concrete from a mixer. When they finish loading, they proceed to their destination. The described model is illustrated using an event diagram as shown in Figure 2-1.

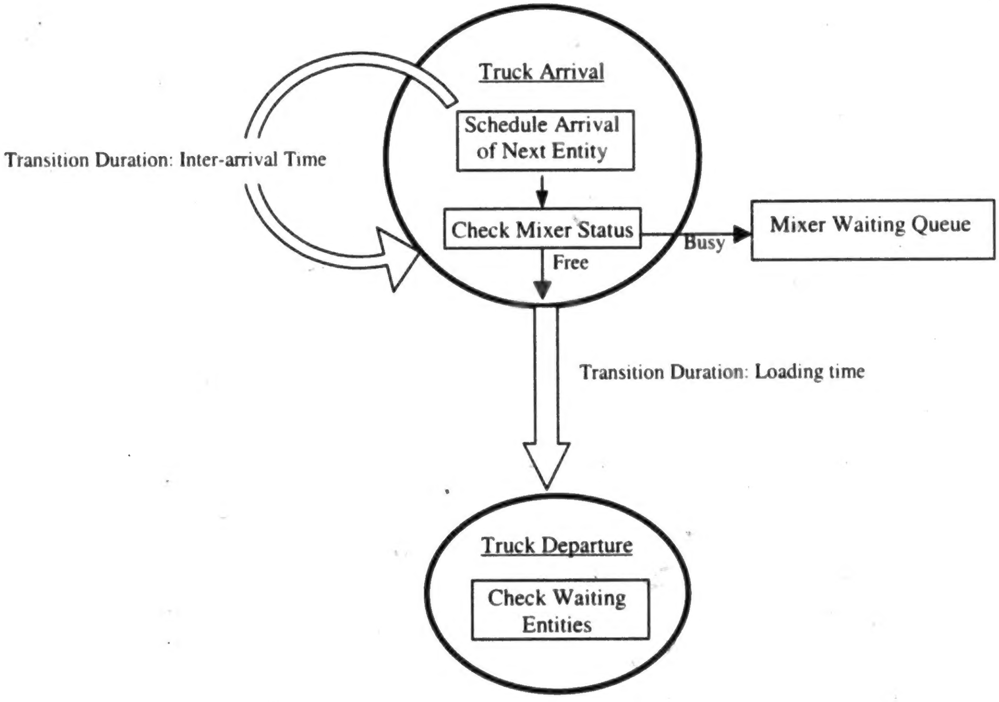

Truck Arrival
Schedule Arrival of Next Entity
Transition Duration: Inter-arrival Time
Check Mixer Status
Busy
Mixer Waiting Queue
Free
Transition Duration: Loading time
Truck Departure
Check Waiting Entities

Figure 2-1 Sample Discrete-Event Model Representation

The "Truck Arrival" event is first added to the event calendar during the initialization of the simulation. When the event is processed as part of the event handler, the arrival of the next truck entity is first scheduled to occur. Then the status of the mixer is checked. If it is busy then the truck entity is added to a queue. Otherwise, the "Truck Departure" event is scheduled. During the processing of the "Truck Departure" event, which occurs when a loading operation has been completed, the status of the waiting queue is checked. If there are any waiting truck entities, then the first one is removed from the queue so that it can start loading. The transition duration between each "Truck Arrival" event is based on the desired inter-arrival time of trucks. The transition duration between the "Truck Arrival" and "Truck Departure" event depends on the desired truck loading time. Table 2-1 explains how the content of the event calendar changes as the above model is processed. The described scenario is based on a truck inter-arrival time of 15 minutes and a loading time of 20 minutes. The last column of the table shows the content of the event calendar in terms of the entity number, event and simulation time. Note how the event calendar always sorts its content according to the scheduled time of the events.

Table 2-1 Sample Discrete-Event Simulation Processing Session

Simulation Time	Current Event	Current Entity	Mixer Status	Queue Content	Event Calendar Content
Initialization	None	None	Free	Empty	(1,Truck Arrive,0)
0	Truck Arrive	1	Busy	Empty	(2,Truck Arrive,15) (1,Truck Depart,20)
15	Truck Arrive	2	Busy	2	(1,Truck Depart,20) (3,Truck Arrive,30)
20	Truck Depart	1	Busy	Empty	(3,Truck Arrive,30) (2,Truck Depart,40)
30	Truck Arrive	3	Busy	3	(2,Truck Depart,40) (4,Truck Arrive,45)
40	Truck Depart	2	Busy	Empty	(4,Truck Arrive,45) (3,Truck Depart,60)
45	Truck Arrive	4	Busy	4	(5,Truck Arrive,60) (3,Truck Depart,60)
60	Truck Arrive	5	Busy	4,5	(3,Truck Depart,60) (6,Truck Arrive,75)
60	Truck Depart	3	Busy	5	(6,Truck Arrive,75) (4,Truck Depart,80)

2.2.2 Object-oriented Simulation

The object-oriented paradigm was applied to simulation in order to produce models that are more comprehensible, modular and extendable. The main difference is that simulation model elements now correspond to their real life counterparts (Ulgen and and Thomasma 1986) and users are no longer required to deal with abstract modeling constructs. The first object-oriented simulation system (OOSS) developed was Simula (Ahl and Nygaard 1966). Simula supported full data encapsulation, inheritance and polymorphism. The original system was based on the ALGOL language. A SmallTalk implementation was later developed (Ulgen and Thomasma 1986). OOSS made it possible to build specialized graphical user interfaces where models can be built using elements that resemble the real world entities (Bischak and Roberts 1991).

SmartSim (Ulgen et. al.1989) demonstrated how object-oriented concepts were used to provide users in the manufacturing industry with a set of elemental simulation operations which can be extended by building new classes based on the basic ones.

OOSS abolished the traditional notion of a single simulation user and introduced the concept of a two user system (Ball and Love 1995). The first user is the simulation developer. The simulation developer is responsible for creating specialized modeling constructs as required. Simulation users build new simulation models by selecting from the available constructs and defining their relationships and parameters.

2.2.3 Modularity and Hierarchical Concepts

Modular modeling is the process of designing "atomic" simulation modules and linking them with other modules developed in the same manner to produce a new model. Modules communicate through well-defined input and output "ports". Further, two modules can be combined to make a new module with its own input and output port. This leads to the concept of hierarchical modeling. These concepts are extremely useful for modeling large scale systems such as construction projects.

Ziegler (1984) was the first to present a formal theory, called DEVS, of how these concepts can be implemented for a discrete event simulation system. A program called PC-Scheme later demonstrated how object-oriented concepts were used to implement the described theory (Zeigler 1987). Luna (1993) further outlined what it means for a simulation system to support hierarchial modeling. Standridge (1995) described a system that utilized modular concepts as part of a network simulation modeling language.

2.2.4 Strategies for Simulation Model Development

Users can typically change the behavior of a simulation model after it is constructed. This is the concept of reusability where the model can be used for a multitude of scenarios. The degree to which users can change the pre-defined simulation behavior is dependent on the development strategy utilized. Simulation systems can generally be classified according to this feature as follows (Ulgen et. al. 1991):

- Fully documented simulation models
- Parameterized simulation models
- Special purpose simulation program generators
- General purpose simulation program generators

With fully documented simulation models, users are required to modify the simulation models by manipulating them at the same level used to originally develop them. This assumes that end users are knowledgeable in the way the simulation system works. Parameterized simulation models allow for model re-use by exposing a set of parameters that users can modify each time the model is simulated. The values of the parameters can be used to modify routing strategies, resource values and entity attributes. With special purpose program generators (SPSPG), users are able to create models by selecting from a list of available domain specific constructs and defining their parameter values as well as their relation to other elements. Examples of such systems include WITNESS and SIMFACTORY (Mathewson 1989), AP2-Earth (Hajjar and AbouRizk 1996), CRUISER (Hajjar and AbouRizk 1998) and CSD (Hajjar, AbouRizk and Xu 1998).

2.2.5 General purpose simulation program generators

GPSPGs are integrated application development frameworks designed to allow expert users to develop, test and deploy domain-specific simulation tools for use by end users. Thomasma (Thomasma and Ulgen 1988) demonstrated how a modular hierarchial framework could be built for the manufacturing industry with support for graphical model development. Other systems developed with the same kind of capabilities include Extend (Karhl 1995), HI-MASS (Fritz et. al. 1995), Create! (Rueger and Behlau 1995), MMS (McKim and Matthews 1996), Arena (Takus and Profozich 1997) and MOOSE (Cubert et. al. 1997).

2.3 Construction Simulation

2.3.1 Methodologies and techniques

Halpin (1977) popularized the use of simulation in construction research with his development of a system called CYCLONE (Cyclic operation network). CYCLONE allowed the user to build models using a set of abstract but simple constructs. The system became the basis for a wide range of construction simulation research efforts with the objective of enhancing the basic system functionality. This included INSIGHT (Paulson 1978), UM-CYCLONE (Ioannou 1989), and RESQUE (Chang and Carr 1987). STROBOSCOPE (Martinez and Ioannou 1994) was another development based on CYCLONE which allowed for dynamic simulations based on the definitions of entity and resource attributes using programming like syntax. DISCO (Huang et. al. 1994) was developed to allow for the use of graphical-based modeling for CYCLONE models.

Although CYCLONE and its derivatives introduced a wider academic audience to computer simulation, its use in the industry was very limited. This was because CYCLONE proved practical only for small-scale applications. Further, the modeling process required simulation training – something that the industry was not generally ready to invest in. As a result, research turned to other concepts that could handle these issues and simplify the modeling process even further.

Chang (1991) introduced the object-oriented concepts to construction simulation modeling. Object orientation improves the readability of simulation systems and produces models that resemble their real life counterparts. In effect, they bridge the gap between physical systems and their computer representation (Oloufa 1993). Their advantage was discussed in detail by Oloufa (1993), who compared a MODSIM based object-oriented implementation of an earth-moving operation to a procedural one. He further concluded that the use of the object-oriented approach leads to reduced coding and improved simulation model readability.

Several researchers applied general simulation concepts to allow for model reusability. Tommelein (1994) and Shi (1997) utilized a library-based modeling approach that allows project simulation models to be assembled from a set of pre-defined components. Oloufa

(1994) developed an object-oriented library-based modeling approach for the construction of parameterized simulation models. Oloufa suggested that the modeling effort would be greatly simplified if pre-assembled "operation libraries" were first constructed and dynamically packaged according to user input obtained through simple forms. This would allow for non-expert users to take advantage of simulation. COOPS (Liu and Ioannou 1992) is an object-oriented implementation of CYCLONE. The authors have since used it as the basics for extending CYCLONE's functionality to provide resource based decision making (Liu and Ioannou 1993).

The concepts of modular modeling were also used to a certain extent. Modular concepts based on those defined by Ziegler (1984) were utilized by Sawhney (1996) to develop large scale simulation systems.

2.3.2 Integration Issues

The first effort to integrate simulation tools with other construction systems can be traced back to the works of Wickard (1989) who presented a system that links simulation activities to CAD model elements. However, the simulation side itself consists of a very basic static model. Another system developed by Touran (1989) demonstrated how rule-based expert systems can be used to assemble library simulation components to create a final model. Other research demonstrated how artificial neural networks (Hajjar, AbouRizk and Mather 1999) and equipment databases (Hajjar and AbouRizk 1996) could be integrated with the modeling environment.

Chapter 3 – Earth-Moving Project Simulation[1]

3.1 Introduction

This chapter presents the main design and implementation steps of a specialized simulation tool called AP2-Earth. The system was developed collaboratively with a local earth-moving contractor. One objective of the system was to introduce the industry to a simulation based analysis for planning earth-moving projects. The other objective was to gather information on the general features that a simulation tool for the heavy civil industry must possess in order to be accepted and successfully used by earth-moving contractors.

This chapter is organized as follows: Section 3.2 provides background information on earth-moving construction. Section 3.3 is a discussion of the main modules of the developed system. Section 3.4 details the structure of the main AP2-Earth module. Section 3.5 presents the results of field case studies performed to verify the system results. The conclusions are then presented in Section 3.6 .

3.2 Background

Earth-moving is a specialized field where large quantities of earth are moved from one location, generally referred to as the cut, to an another location, referred to as the fill.

---

[1] A version of this chapter was published as "Building a Special Purpose Simulation Tool for Earth-moving Operations" in the proceedings of the 1996 Winter Simulation Conference, ASCE, pp. 1313-1320.

Examples of such projects include overburden removal for mining operations and construction of earth filled dams.

Earth-moving projects consist of many interacting processes including preparation, loading, hauling, dumping and spreading. Preparation is done if the earth is not suitable for immediate loading and requires ripping. Loading is the process of transporting earth from the prepared earth pile into incoming trucks. This is done using loaders, shovels or backhoes. Hauling involves trucks traveling through roads with varying slopes and ground conditions as well as traffic intersections in order to transport earth and return. Dumping is the transfer of earth from the trucks into a spreading pile. This pile is spread by a number of dozers as part of the spreading process.

The most important element of preparing a cost estimate is the determination of the number of production units to use. To obtain the number and type of production units, construction estimators rely on the client's specifications to arrive at certain required data. This includes the following information:

- The overall site layout, including the location of the source and placement areas.
- Overall quantities to be prepared, hauled and placed.
- Properties of the haul paths, including distances, grades and rolling resistance.

The first step is to decide on the type of excavator to be used. This depends on equipment availability, type and quantity of the soil to be moved, and the schedule requirements of the project. After the excavator is chosen, the type of trucks to be used is determined. This too depends on equipment availability, type of excavator and haul path properties. Next, the cycle time of the trucks is determined. The elements of a complete truck cycle are:

- Loading time
- Travel loaded time
- Dump time
- Travel empty time

Loading time is calculated using the excavator's expected production rate plus any allowance for waiting and positioning. The travel loaded and travel empty duration are obtained by first dividing the haul paths into many segments, each with a relatively uniform grade and rolling resistance. This information, along with equipment manufacturer specifications, is used to determine the expected travel speed of the trucks across each road segments. Travel speed along with the length of each segment is used to arrive at the travel duration. Dump time is determined using historical information.

The truck cycle time along with the truck capacity is used to calculate the number of trucks required to match the production capability of the excavator. Further, extra trucks are generally made available on site in case of breakdown and as a replacement for trucks undergoing maintenance. The described estimating procedure is adequate for simple cases where single source and placement areas are present and a single type of truck is used. This process becomes time consuming and tedious in the case of multiple source and placement locations, complicated haul paths with many changes in total resistance, and varying types of trucks. Further, estimators cannot accurately determine the waiting duration associated with excavator usage or as a result of fleet interactions between multiple paths.

3.3 The Overall System Structure

AP2-Earth was developed to address the stated issues and to simplify the tender preparation process on large and complex earth-moving projects. It was determined that to ensure its acceptance, AP2-Earth would have to be built with the capability to integrate with other modules such as equipment inventory databases and estimating programs. These supporting modules were also developed and provided with the main simulation module. A graphical depiction of the overall AP2-Earth modules and how they relate to each other is illustrated in Figure 3-1.

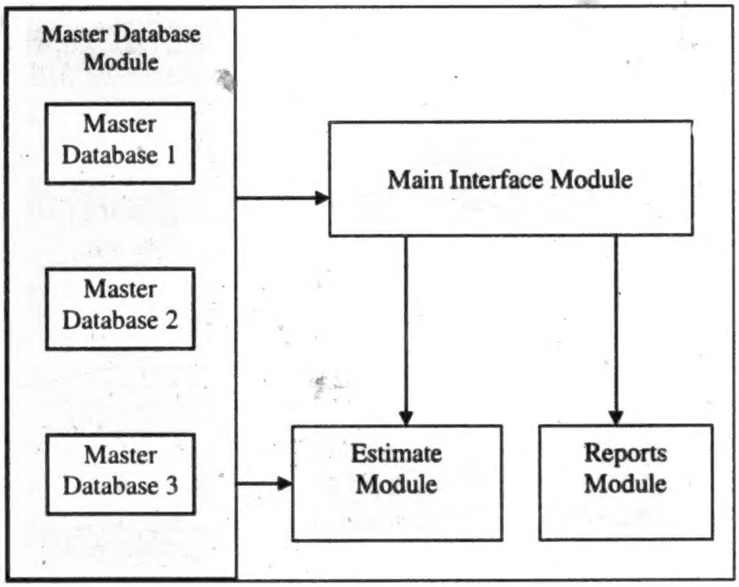

Master Database Module
Master Database 1
Master Database 2
Master Database 3
Main Interface Module
Estimate Module
Reports Module

Figure 3-1 AP2-Earth Modules

The master database module consists of standard equipment and labour specifications that are available for use by the contractor. The database stores various equipment properties, some of which can be used directly in the simulation environment or at a later stage for estimation purposes. The main interface module is where simulation modules are built. Equipment data can be imported from the master database modules. Two other modules are used to examine the simulation results. The first is the reports module, which provides various types of information ranging from high level production reports to detailed equipment performance and utilization reports. The other is the estimate module, which allows users to develop estimates and link certain inputs directly to the simulation results. The next section will detail the development features of the main interface module.

3.4 Design And Development Of Ap2-Earth

3.4.1 Overview

The first step in the development process consists of the definition of the project level classes shown in Figure 3-2. These classes encapsulate behaviors, properties and methods for the representation and manipulation of those processes generally found on earth-moving projects (eg. preparation, hauling). Similarly, in the second step, classes are defined for the representation and manipulation of process level objects (eg. road segments, intersections).

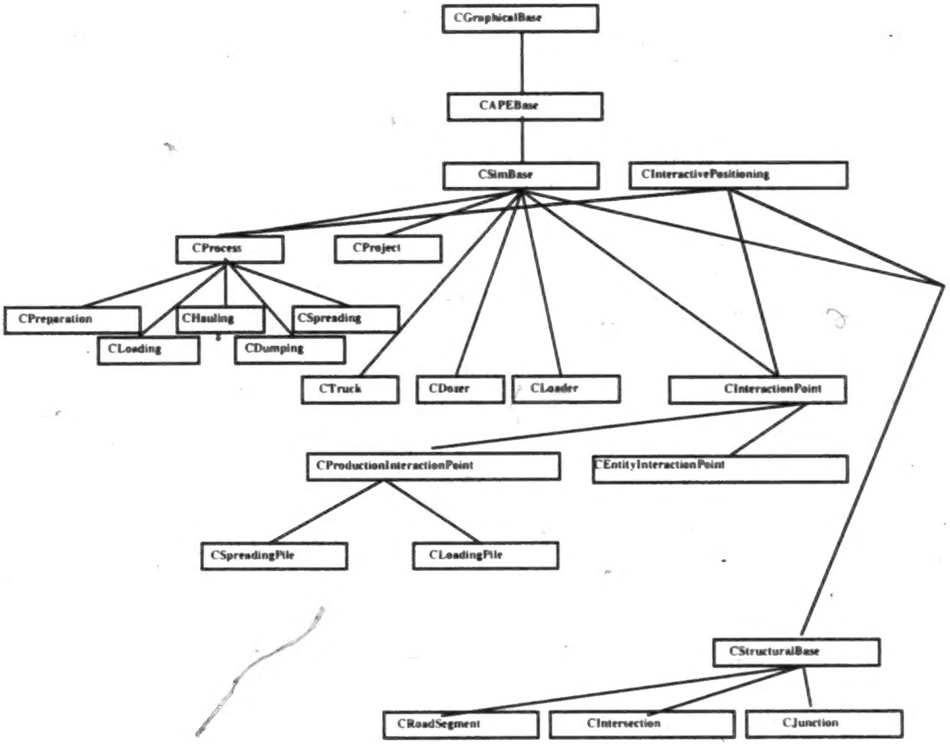

CGraphicalBase
CAPEBase
CSimBase
CInteractivePositioning
CProcess
CProject
CPreparation
CHauling
CSpreading
CLoading
CDumping
CTruck
CDozer
CLoader
CInteractionPoint
CProductionInteractionPoint
CEntityInteractionPoint
CSpreadingPile
CLoadingPile
CStructuralBase
CRoadSegment
CIntersection
CJunction

Figure 3-2 AP2-Earth Class Hierarchy

The third step involves the definition of the interaction points, which are used to link the processes. For example, the preparation process can be linked to a loading process through the preparation pile; this adds a dependency between the two processes whereby loading production cannot exceed that of preparation. Table 3-1 provides a brief overview of a selected number of classes.

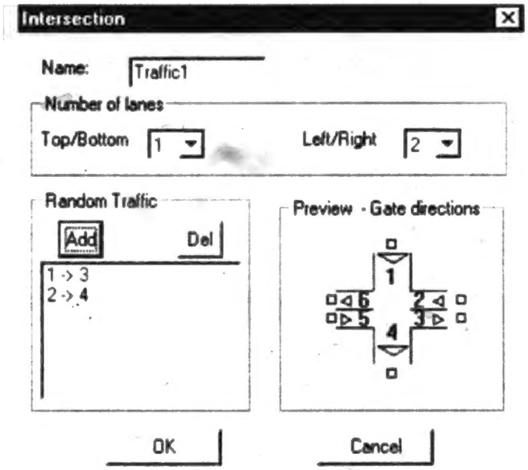
Table 3-1 Description of AP2-Earth Classes

Class	Level	Functionality	Comment
CGraphical-Base	Project	Basic graphical state and position representation/ manipulation functionality.	This class aids in satisfying the objective of graphical manipulation by allowing objects to posses visual editing capabilities.
CInteractive-Positioning	Project/ Process	"click and drag" interactive positioning functionality	This added behavior simplifies data entry for position and linking information of all classes.
CSimBase	Project	Simulation behavior support	This class provides various "user hooks" that allow the specialized classes to perform custom manipulation of numerous behaviors.
CProcess	Project	Basic Process Definition	A CProcess class defines basic functionality of a high level process such as ability to connect to interaction points and the base structures needed to define a simulation model. Child classes (i.e. CPreparation, CHauling, Cspreading,…) implement the specific simulation models.
CProject	Project	Project level information and resources	Project level information includes the list of all processes, interaction point definitions, as well as any global resources that are available to all the processes.
CStructural-Base	Project/ Process	Connection management	Classes like CRoadSegment and CIntersections, which constitute the structural elements of the model, derive from this class.
CRoad-Segment	Process	One way uniform road segment modeling	Information such as grade, rolling resistance and velocities is maintained by this class
CIntersection	Process	Traffic Intersection modeling	Supports the modeling of interactions at a traffic intersection. Also provides capabilities for modeling external traffic processes such as highway traffic
CInteraction-Point	Project	Inter-Process Interaction Management	Allows two process to be linked in order to transfer entities or add dependencies

In general, an AP2-Earth object (i.e. an instance of a non-abstract class, typically at the lowest level of the hierarchy) encapsulates a variety of information and methods depending on the inheritance branch. The main supported behaviors are: graphical, simulation, data collection, statistical analysis, and reporting.

3.4.2 Graphical Behavior

Graphical representation and manipulation functionality is gained by deriving from the CGraphicalBase class. Child classes are required to implement certain virtual functions in order to conform to the overall visual manipulation objective. Each class typically has an associated graphical dialog box where the user can enter and change information in a very intuitive manner. Figure 3-3 shows the CIntersection dialog box, which allows users to specify attributes such as number of lanes and direction of travel along each gate. Figure 3-4 shows the CTruck dialog box where truck properties such as capacity, dumping duration and quantity are defined. The graphical behavior of the CProject class is, in fact, the main program screen where AP2-Earth objects can be added, deleted or linked. This screen is shown in Figure 3-5.



Name: Traffic1

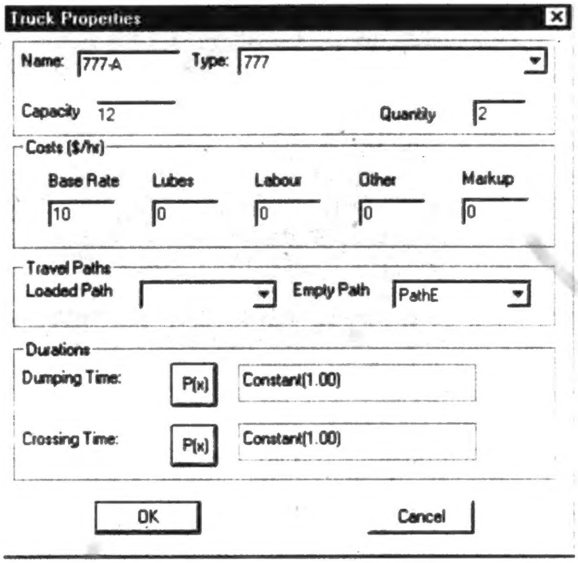
Number of lanes

Top/Bottom 1

Left/Right 2

1 -> 3
2 -> 4

Figure 3-3 CIntersection Graphical Behavior


Truck Properties
OK
Cancel

Name: 777-A

Type: 777

Capacity 12

Quantity 2

Costs ($/hr)

Base Rate	Lubes	Labour	Other	Markup
10	0	0	0	0

Travel Paths

Loaded Path:

Empty Path: PathE

Durations

Dumping Time: P(x) Constant(1.00)

Crossing Time: P(x) Constant(1.00)

Figure 3-4 CTruck Graphical Behavior

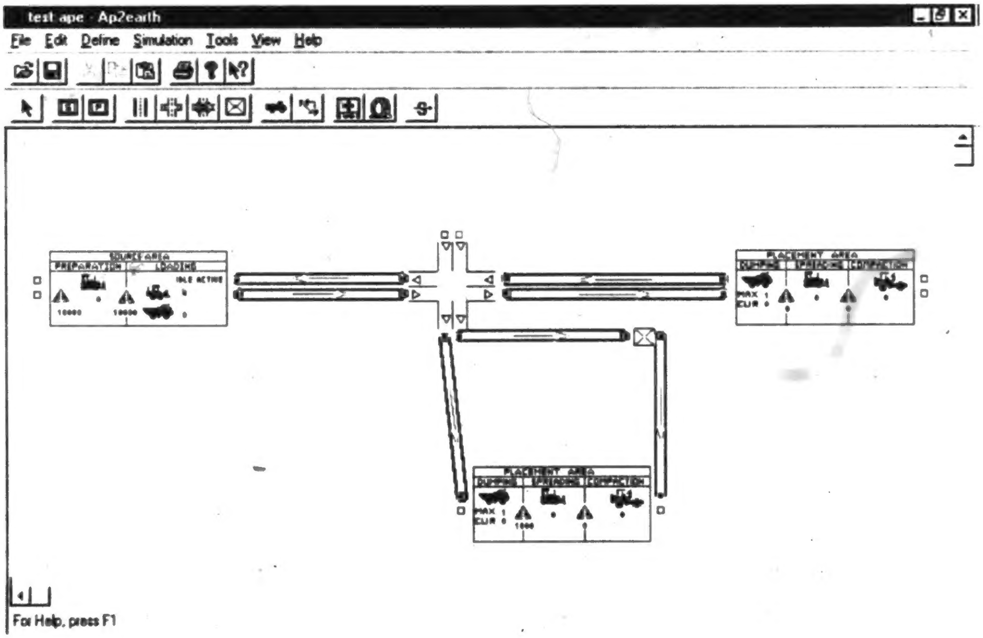

test.ape - Ap2earth
File
Edit
Define
Simulation
Tools
View
Help
SOURCE AREA
PREPARATION
LOADING
IDLE ACTIVE
10000
10000
PLACEMENT AREA
DUMPING
SPREADING
COMPACTION
MAX 1
CUR 0
PLACEMENT AREA
DUMPING
SPREADING
COMPACTION
MAX 1
CUR 0
1000
For Help, press F1

Figure 3-5 CProject Graphical Behavior

3.4.3 Simulation Behavior

Each class derived from CSimBase defines a unique simulation model and becomes an independent entity. A simulation model consists of a set of instructions used to represent the sequence of activities that typically take place in the real life scenario. This set of instructions includes calls to an event scheduling subsystem, requests for resources such as loaders and preparation piles as well as statistics collection routines. Detailed information on the simulation models is presented later in this chapter.

3.4.4 Data Collection Behavior

Each AP2-Earth class is responsible for tracking any data required for the analysis of the element it represents. For example, a CTruck class is typically responsible for collecting data on: cycle duration, waiting duration at the loader and dump, interaction durations, and maintenance duration. CProcess derived classes are mainly concerned with tracking overall production data. Each class has a related table in a relational database where tracked data can be stored for later analysis.

3.4.5 Statistical Analysis Behavior

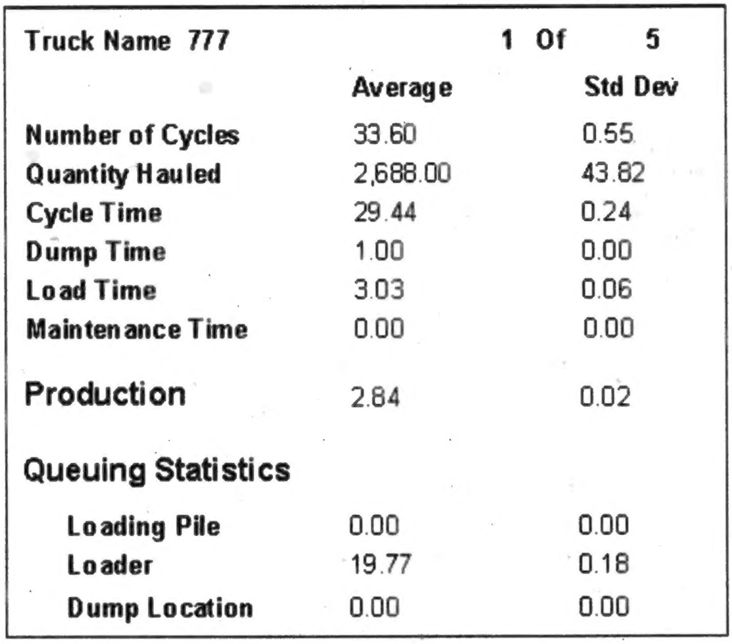
Classes are given the opportunity to perform post-run (in the case of a multi-run simulation) or post simulation statistical analysis. This is done by overriding the two functions, PostSimulationRun() and PostSimulation() of the CSimBase class. In the current implementation of AP2-Earth, the classes simply pass a query to the relational database, which handles the analysis. An example query used by the CProcessSpreading is shown in Figure 3-6. The database subsystem executes the query and returns the results to the calling class.

```
SELECT iteration, PlacementCode, Avg(SpreadingDuration) , Avg(SpreadingAmount)
        Avg([SpreadingAmount]/[SpreadingDuration]) AS Production
FROM SimProcessSpreading
GROUP BY iteration, PlacementCode;
```

Figure 3-6 CTruck Data Analysis Behavior

3.4.6 Reporting Behavior

AP2-Earth Classes requiring the production of simulation reports override the CSimBase function CSimulationReport(). The classes then select which statistical indicators to display and send a report definition command to an independent reports module. This brings up a report for the user to see and print if desired. An example report generated by the CTruck class is shown in Figure 3-7.



```
Truck Name  777                        1  Of      5
                          Average            Std Dev
Number of Cycles          33.60              0.55
Quantity Hauled           2,688.00           43.82
Cycle Time                29.44              0.24
Dump Time                 1.00               0.00
Load Time                 3.03               0.06
Maintenance Time          0.00               0.00

Production                2.84               0.02

Queuing Statistics

   Loading Pile           0.00               0.00
   Loader                 19.77              0.18
   Dump Location          0.00               0.00
```

Figure 3-7 CTruck Reporting Behavior

3.4.7 Simulation Behavior

3.4.7.1 Overview

As explained earlier, the simulation models are implemented as a behavior of the classes deriving from the CSimBase class. The development of the simulation models begins by a preliminary conceptual design of each process followed by the development of a formal simulation model. The next step is the development of the interaction points which link processes together. The following section details how the CHauling class implements the simulation behavior.

3.4.7.2 Preliminary Conceptual Definition

The hauling process involves a number of trucks hauling earth from the preparation area or "source" to the destination or "placement". Roads are normally modeled as road segments with properties such as grade, length and maximum travel speed. The haul route could contain various intersections and junctions where trucks must stop and give way to other traffic that might have higher priority. The trucks follow a static path from the source to the placement known as the "travel loaded" path and return on the "travel empty" path. The source and placement in both paths are the same, therefore trucks always follow a closed loop. Different trucks could follow different paths since earth-moving operations may involve several sources and placements.

The travel times of trucks can be obtained from empirical travel time curves such as the one shown in Figure 3-8. These curves give approximate travel duration given the length and total resistance of the road segment for both loaded and empty trucks. Total resistance is composed of grade resistance and rolling resistance. Grade resistance is a measure of the force that must be overcome to move trucks over uphill slopes. Rolling resistance is a measure of the force that must be overcome to roll or pull a wheel off the ground.

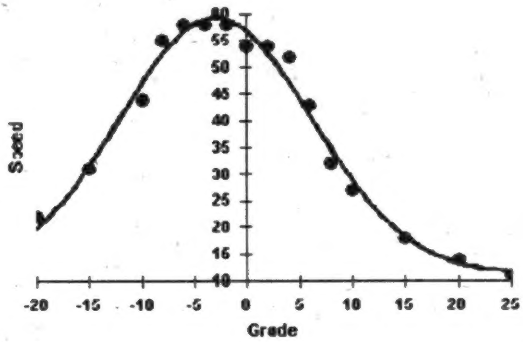


Figure 3-8 Sample Travel time Curve

Within earth-moving projects, intersections serve to regulate the traffic between arriving entities. Trucks must stop at intersections and check before proceeding. Intersections are not simple resources since the maximum number of trucks or other types of traffic allowed at any one time is variable and dependent on the "crossing path" of entities. As illustrated in Figure 3-9, Case 1 presents no conflict between the crossing behavior of incoming traffic. The intersection serves as a relay station and no waiting is involved. In Case 2 however, traffic could queue at A or B since the desired paths cross.

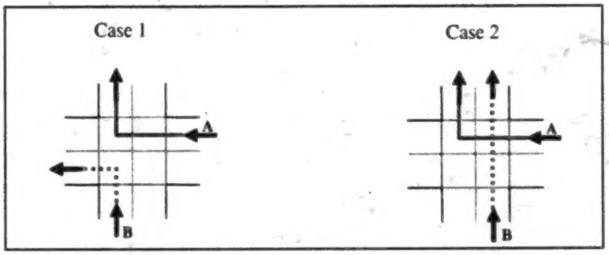


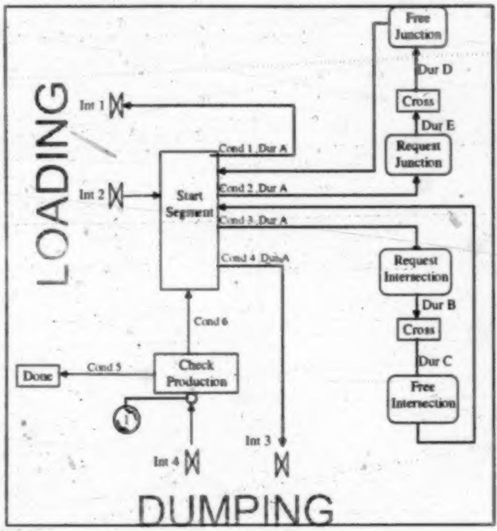
Figure 3-9 Example Intersection Crossing Scenarios

3.4.7.3 Simulation Level Design

The simulation level design is performed by constructing event flow diagrams using the symbols illustrated in Table 3-2. These diagrams define the event sequence of each element and can be directly translated into discrete-event simulation models. The hauling event flow diagram is shown in Figure 3-10(a). The corresponding set of discrete event simulation calls in pseudo code format is shown in Figure 3-10 (b).

Table 3-2 Event Flow Diagramming Legend

Symbol	Description
Event	Represents a simulation event or transition in simulation.
Cond A → Dur A	Represents an activity with a defined duration and a criteria in the case of conditional branching.
Resource	Allocated or releases a certain resource.
	Accumulate a production level for any desired purpose. Used to track the cumulative production level.
+(Int. 1	Represents production interaction points.
→⋈ Int. 1	Represents an entity interaction point. The arrow indicates whether entities are arriving at the process or exiting from it.
*→	Indicates the starting event of a process.


Free Junction
Dur D
Cross
Dur E
Request Junction
LOADING
Int 1
Cond 1 ,Dur A
Int 2
Start Segment
Cond 2 ,Dur A
Cond 3 ,Dur A
Cond 4 ,DurA
Request Intersection
Dur B
Cross
Dur C
Free Intersection
Cond 6
Done
Cond 5
Check Production
1
Int 3
Int 4
DUMPING

```
Get next event from event queue
if event = START_SEGMENT then
      Set TravelTime to predicted travel duration
      If travel loaded path is complete then
            Transfer truck into dumping process
      else if travel empty (return) path is complete then
            Transfer truck into loading process
      else if next object is an intersection then
            Schedule REQUEST_INTERSECTION ,
else if event = REQUEST_INTERSECTION
      Wait until intersection resource is available
      schedule FREE_INTERSECTION ,CrossingTime
else if event = FREE_INTERSECTION
      Release intersection resource
         schedule START_SEGMENT
```

Figure 3-10 Hauling Simulation Model

As explained before, the traffic behavior at intersections depends on several factors including priority, "crossing path" and crossing duration of incoming traffic. This behavior is modeled using a state grid. The grid determines whether a certain traffic block is currently busy or free. Traffic entities arriving at an intersection cross only when all the blocks constituting the crossing path are free; otherwise, they will queue. When an entity crosses the intersection and releases the blocks, waiting entities are allowed to proceed based on priority. If priorities are equal, they are served on first-come-first-serve basis. The result is a sort of four-way intersection.

3.4.7.4 Interaction Design

Once the definition of the other processes is complete, they are linked to form a representative model of the whole project. Linking is achieved by deriving from the CInteractionPoint class. In earth-moving, two types of interactions are present:

production-based and entity-based. An example of a production-based interaction is between CLoading and CPreparation. On one side, dozers in the preparation area continually increase the level in the pile. On the other side, trucks are being loaded using that same pile. Queuing is possible only from the loading side as trucks will wait if not enough earth is present. This behavior is implemented by the CLoadingPile class. Entity based interaction points are present between loading and hauling, and between hauling and dumping. The entities that are being transferred between the two processes represent trucks. The CEntityInteractionPoint class implements this behavior. An example project is shown in Figure 3-11. It consists of several possible preparation processes, loading, hauling, dumping, and spreading processes.

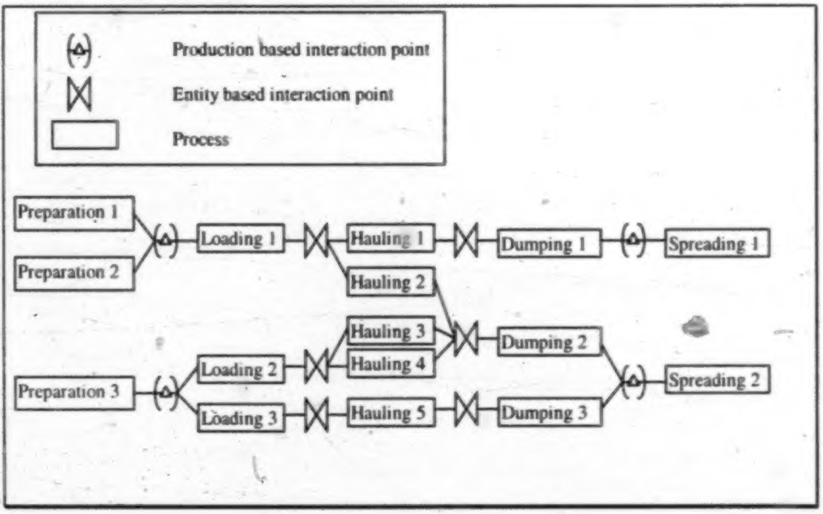

Production based interaction point
Entity based interaction point
Process
Preparation 1
Preparation 2
Loading 1
Hauling 1
Hauling 2
Dumping 1
Spreading 1
Hauling 3
Hauling 4
Dumping 2
Loading 2
Preparation 3
Spreading 2
Loading 3
Hauling 5
Dumping 3

Figure 3-11 Example Earth-Moving Project

3.4.7.5 Synchronization of Process Models

The simulation behavior of the various processes and the interaction points have been designed to run as independent models. However, a mechanism is still required to synchronize the various simulation events. This task is the responsibility of the CProject's simulation model. Figure 3-12 illustrates how this task is done. Mainly, events generated by the individual processes are scheduled in a single event list. The next event routine used by the simulation processor removes the next event and routes it back to the appropriate process based on an embedded attribute.

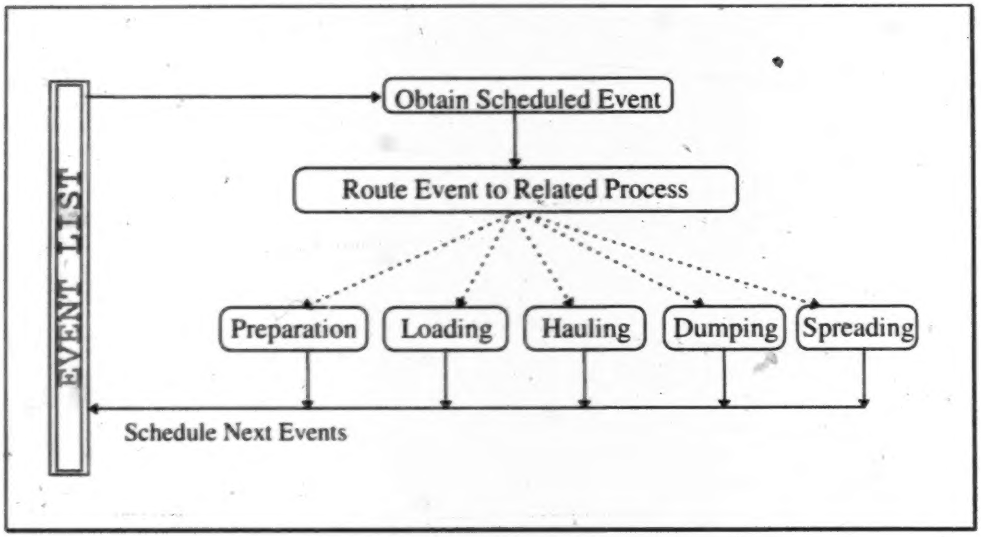

EVENT LIST
Obtain Scheduled Event
Route Event to Related Process
Preparation
Loading
Hauling
Dumping
Spreading
Schedule Next Events

Figure 3-12 CProject Simulation Model

3.4.8 Implementation Platform and Model Building Steps

AP2-Earth was implemented using the Visual C++ language, which allowed the system to utilize the C++ object-oriented language and run under the graphical Microsoft Windows environment. The reports module was constructed using Microsoft Access. This allowed the simulation results to be manipulated using database queries and presented as customizable database reports. The model building and analysis steps are as follows:

1) Construct project layout using interface

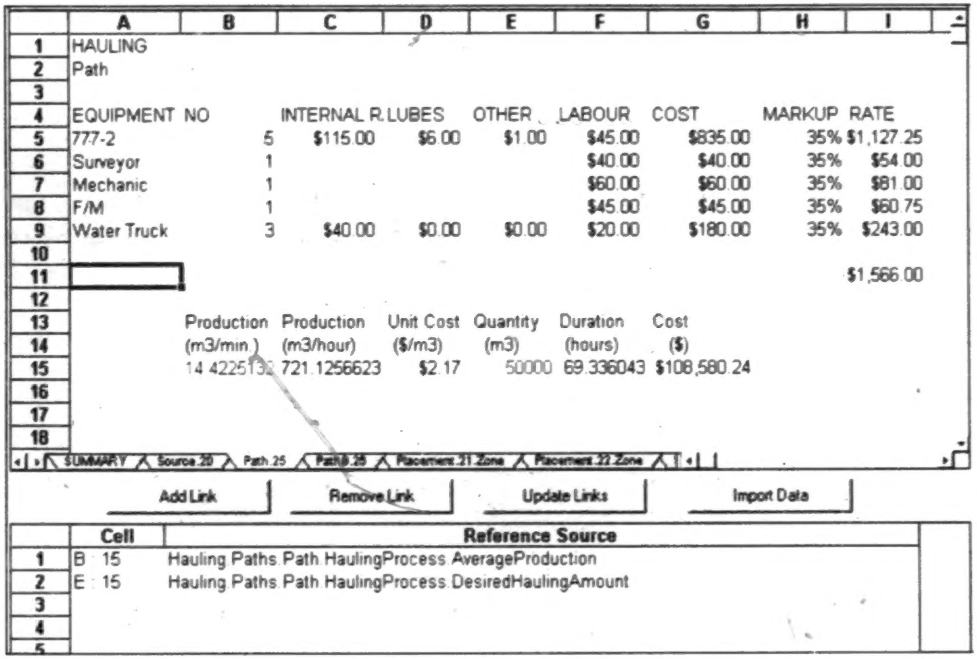
- Place sources and placements.
- Build road layout using road, connector, intersection and junction objects.
- Add random traffic processes at desired intersections.

2) Create paths
   - Create one or more "Travel Loaded" paths and assign hauling quantities.
   - Create one or more "Travel Empty" paths.
3) Create Truck(s)
   - Create at least one truck and define the properties
4) Define Source(s) properties.
   - Create at least one loader and define loading durations with truck(s).
   - Define amount prepared and amount to prepare.
   - If preparation is required, create dozers.
5) Define Placement(s) properties
   - Define amount to spread
   - If spreading is required, create dozers
   - Define amount to compact
   - If compacting required, define compactors.
6) Define breakdowns if desired
7) Specify Simulation Options
   - Number of iterations
   - Termination condition
   - Traffic Priority
8) Simulate project and examine results in the reports module

3.4.9 Design Notes

The approach used in AP2-Earth provides numerous benefits. First, the graphical editing environment gives the tool a user-friendly interface for the definition and manipulation of complex earth-moving projects. Objects can be created and manipulated using simple "click and drag" mouse operations. Users do not need to posses any simulation knowledge in order to use the tool. Second, the simulation modeling constructs are extremely flexible and capable of representing a large number of earth-moving scenarios. Third, due to independence of AP2-Earth objects and the defined interaction points, the tool is highly scaleable and is capable of representing any number and combination of processes such as preparation and spreading. Fourth, the encapsulation concept allows the tool to be easily extended since the addition of new modules does not require any changes to be done to the existing classes. For example, a compaction module has recently been added by defining the CProcessCompaction class and a linking class to interface it with CProcessSpreading.

Another added benefit is the capability to export data in a variety of formats. For each required destination module, an export behavior can be added to the AP2-Earth classes. Examples of destination modules include historical databases, estimate packages and accounting system. Figure 3-13 presents an estimate module where the information contained was automatically imported from AP2-Earth simulation results. The top section contains a spreadsheet where crews and costs are listed. The bottom section lists current linking information between certain spreadsheet cells and properties of AP2-Earth objects.



	A	B	C	D	E	F	G	H	I
1	HAULING								
2	Path								
3									
4	EQUIPMENT	NO	INTERNAL R.	LUBES	OTHER	LABOUR	COST	MARKUP	RATE
5	777-2	5	$115.00	$6.00	$1.00	$45.00	$835.00	35%	$1,127.25
6	Surveyor	1				$40.00	$40.00	35%	$54.00
7	Mechanic	1				$60.00	$60.00	35%	$81.00
8	F/M	1				$45.00	$45.00	35%	$60.75
9	Water Truck	3	$40.00	$0.00	$0.00	$20.00	$180.00	35%	$243.00
10									
11									$1,566.00
12									
13		Production	Production	Unit Cost	Quantity	Duration	Cost		
14		(m3/min.)	(m3/hour)	($/m3)	(m3)	(hours)	($)		
15		14.4225132	721.1256623	$2.17	50000	69.336043	$108,580.24		
16									
17									
18									

	Cell	Reference Source
1	B : 15	Hauling.Paths.Path.HaulingProcess.AverageProduction
2	E : 15	Hauling.Paths.Path.HaulingProcess.DesiredHaulingAmount
3		
4		
5		

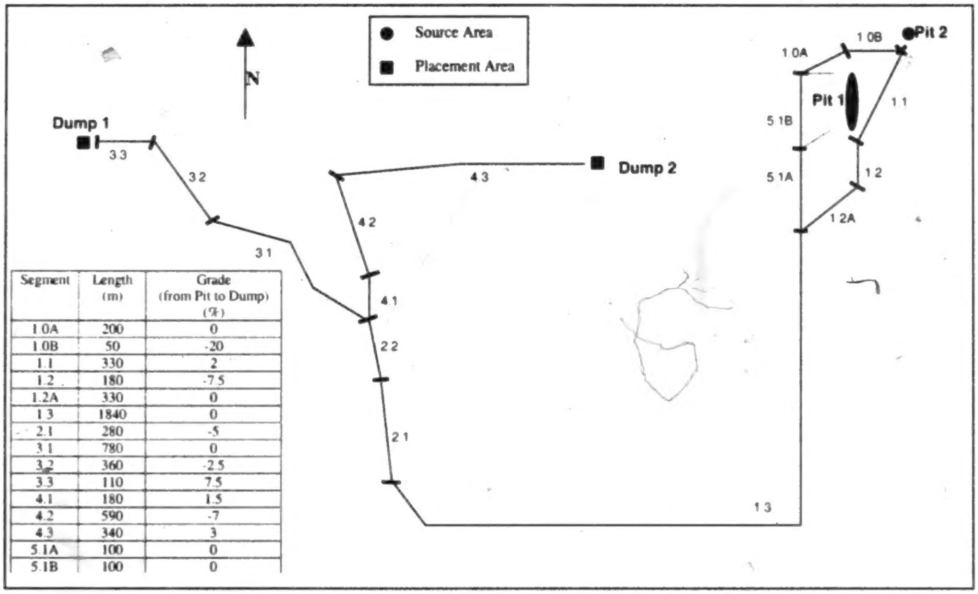
Figure 3-13 Integration with Estimate Module

3.5 Case Study

On March 14 and 15, 1997, truck and excavator performance data was collected from the Syncrude mine site for comparison with AP2-Earth predictions. The objective of this study was to verify the model's predicted parameters and examine whether the tool can be used in a real life setting.

The project required the clearing of large volumes of overburden material. Two excavators at a time were used and the work was performed in two layers or "benches". One excavator worked on a top layer of approximately 5 meter and another followed behind on the lower bench which was approximately 10 meters in height. Although the site topography varied over the span of the project, the elevations and distances were relatively uniform during the two-day data collection period. The site layout is shown in

Figure 3-14. The two pits shown represent the two benches. The material was hauled to two dump locations approximately the same distance from the pits. Site information including elevations and lengths were determined using a GPS survey of the haul roads. The results are shown in a table on the left of the diagram.



Segment	Length (m)	Grade (from Pit to Dump) (%)
1.0A	200	0
1.0B	50	-20
1.1	330	2
1.2	180	-7.5
1.2A	330	0
1.3	1840	0
2.1	280	-5
3.1	780	0
3.2	360	-2.5
3.3	110	7.5
4.1	180	1.5
4.2	590	-7
4.3	340	3
5.1A	100	0
5.1B	100	0

Figure 3-14 Overburden Clearing Project Site Layout

3.5.1 Excavators

Three types of excavators were used on this project: ONK front shovel, Caterpillar EX-1800 backhoe and Caterpillar EX-3500 front shovel. The ONK and the EX-1800 were operating on the first day. On the following day, the ONK and the EX-3500 were used. An important input to the simulation model is the loading duration of the excavator with each type of truck. Due to the small sample size collected, the truck types were ignored and the entire sample was analyzed as a single set. The excavator's production, measured in loose cubic meter per hour, was calculated from the study data. This production is considered ideal since it does not include any waiting times or delays. For the simulation model, the ideal production is used in combination with the truck capacities to obtain the required loading duration. A summary of the observed ideal productions is provided in Table 3-3.

Table 3-3 Observed Ideal Production of Excavators

	Day 1		Day 2	
Excavator	Location	Production (LCM/Hr)	Location	Production (LCM/Hr)
EX-1800	Top Bench	925		
ONK	Bottom Bench	832	Top Bench	1119
EX-3500			Bottom Bench	1304

3.5.2 Trucks

Four different types of trucks were used on this project. Each type has distinct characteristics that determines its maximum capacity, travel time and dumping time. Table 3-4 shows the features of each truck type used. Capacity, given in loose cubic metres, was estimated by the field superintendent and verified using manufacturer data. The initial intent was to record travel time across each road section for each truck type. However, this was not practical due to time and safety concerns. Instead, the overall round trip time of each truck type was recorded and this value was compared to the equivalent value predicted by AP2-Earth.

Table 3-4 Truck Characteristics

Type	Abbreviation	Capacity (Tonnes)	Capacity (LCM)
Caterpillar 785	C785	150	60
Caterpillar 789B	C789	200	75

Titan 220	T220	200	85
Titan 240	T240	220	97

The truck dumping activity was observed and data was collected over a period of 30 minutes. The results are shown as a histogram in Figure 3-15. A visual inspection of the histogram indicates that a normal distribution is a good representation of the data. As a result, the dumping time is assumed to be normal with a mean of 1.42 minutes and a standard deviation of 0.21.

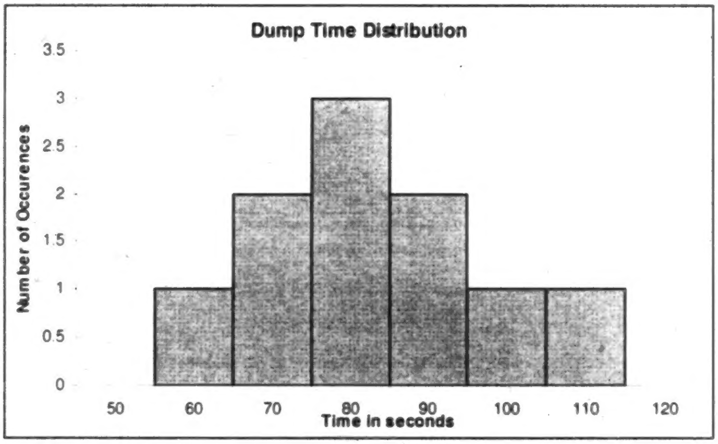


Figure 3-15 Observed Dumping Time

3.5.3 Fleet Interactions

Inter-fleet interactions occur at various locations throughout the site. Understanding and modeling such interactions is vital for the accurate prediction of the travel time. Figure 3-16 illustrates the traffic flows around the loading areas. The highlighted interaction zones indicate points where truck paths cross and delay occasionally incurred. The amount of time lost on each cycle was not recorded because it was observed to be negligible during the study period.

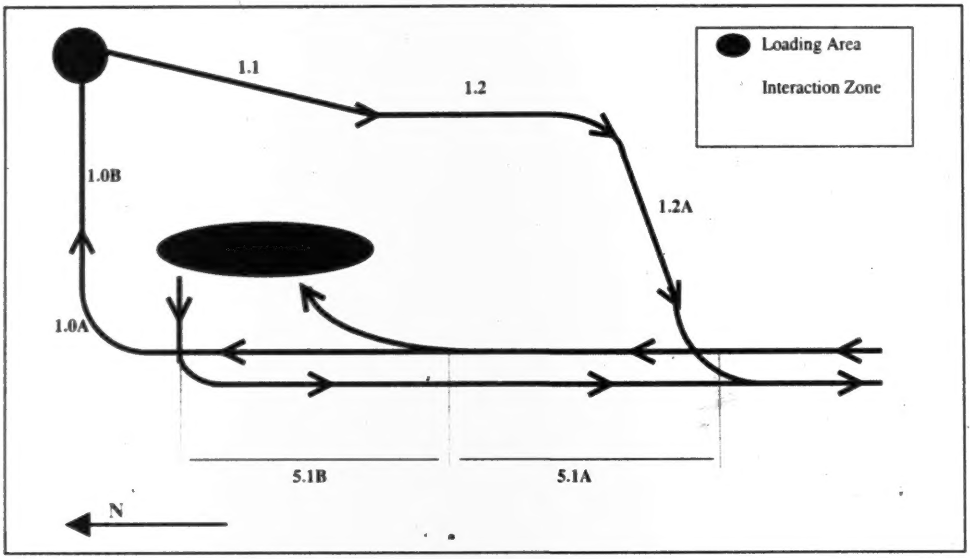


Figure 3-16 Traffic Flow around Loading Area

3.5.4 Model Layout

An AP2-Earth simulation model was constructed as shown in Figure 3-17. The two source objects on the left side represent the pit locations. On the right, two placement areas are used to represent the dump locations. The shown intersections were used to model the possible interactions. The left side of the figure is a representation of the layout shown in Figure 3-16. The three intersections model the interaction zones discussed. Note the similarity between the conceptual model representation and the final simulation model layout.

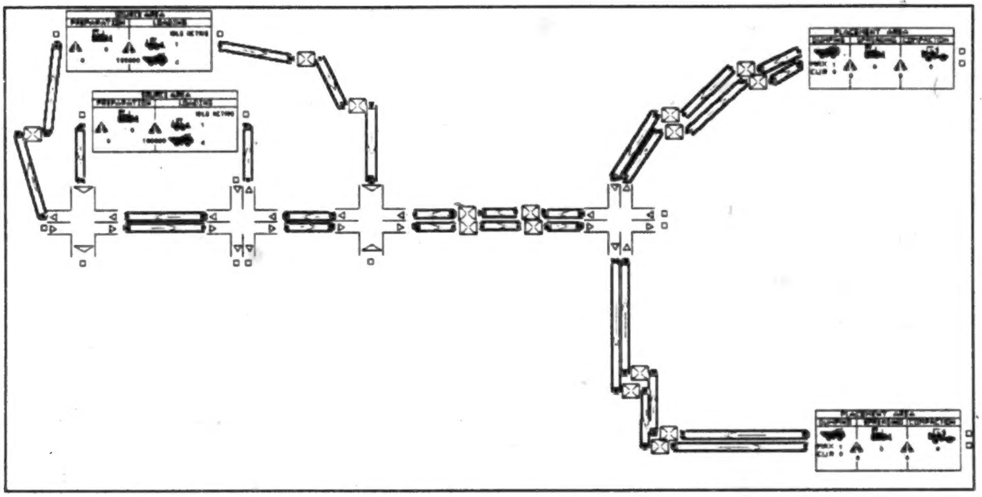


Figure 3-17 AP2-Earth Model Representation

3.5.5 Scenario 1

The first modeled scenario represents the situation that existed during the first day of the study. The EX-1800, located in Pit 2, and the ONK, located in Pit1, along with two truck fleets were operating. The first truck fleet consisted of one C785, one C789 and one T240, and traveled between Pit 1 and Dump 1. The second truck fleet, consisting of one C789 and three T220's, traveled between Pit 2 and Dump 2. Priority at intersections and source areas is set to "First Come First Serve". The model was simulated for 2000 minutes, which allowed the system production to reach steady state. The results of the simulation are summarized and compared to actual data in Table 3-5, Table 3-6, and Table 3-7.

Table 3-5 Comparison of Actual and Predicted Travel Durations for Scenario 1

		Field Observations			**AP2-Earth Simulation Results**		
Fleet	**Truck Type**	**# of obs.**	**Average (min)**	**Std.Dev. (min)**	**Travel Time**	**Delay at Intersections**	**Interactions**

					(min)	(sec/cycle)	
1	C789	3	12.63	0.46	12.83	3	0.00
1	C785	1	11.75	0.00	13.07	6	0.00
1	T240	2	12.65	0.65	12.65	4.2	0.00
2	C789	1	15.32	0.00	14.13	6	0.00
2	CT220	5	14.96	0.88	14.43	4.8	0.00

Table 3-6 Comparison of Actual and Predicted Waiting Duration for Scenario 1

		Field Observations			AP2-Earth Simulation Results (in minutes)
Fleet	Truck Type	# of obs.	Average (min)	Std.Dev. (min)	
1	C789	4	2.61	0.94	2.56
1	C785	3	5.51	0.84	3.33
1	T240	2	2.02	1.34	1.25
2	C789	2	1.99	2.33	1.39
2	CT220	6	0.63	0.57	0.26

Table 3-7 Comparison of Actual and Predicted System Productions for Scenario 1

		Field Observation	AP2-Earth Simulation Results	Comparison	
Exc.	Ideal Production (LCM/Hr)	Production (LCM/Hr)	Production (LCM/Hr)	Deviation	Percent Error
ONK	832	789	832	43	5.4%
EX-1800	925	880	918	38	4.3%

3.5.6 Scenario 2

The second modeled scenario represents the situation present during the second day of the study. On this day, the EX-3500 was operating in Pit 1, and the ONK in Pit 2. Similar to the first day, two truck fleets were used for hauling overburden. The first truck fleet consisted of one C789, one T220, and two T240's; they traveled between Pit 1 and Dump 1. The second truck fleet, consisting of one C785 and three T220's, traveled between Pit 2 and Dump 2. Priority at intersections and source areas is also set to "First

Come First Serve". The model was simulated with the new information for 2000 minutes. The results and comparisons are shown in Table 3-8, Table 3-9, and Table 3-10.

Table 3-8 Comparison of Actual and Predicted Travel Durations for Scenario 2

		Field Observations			AP2-Earth Simulation Results		
Fleet	**Truck Type**	**# of obs.**	**Avg. (min)**	**Std.Dev. (min)**	**Travel Time (min)**	**Delay at Intersections (sec/cycle)**	**Interactions**
1	C789	3	13.80	0.35	12.83	4.2	0.00
1	T220	4	13.43	0.56	13.07	3.6	0.00
1	T240	8	12.82	0.50	12.65	4.2	0.00
2	C785	1	16.50	0.00	14.13	1.8	0.00
2	CT220	7	14.74	0.67	14.43	2.1	0.00

Table 3-9 Comparison of Actual and Predicted Waiting Duration for Scenario 2

		Field Observations			AP2-Earth Simulation Results (in minutes)
Fleet	**Truck Type**	**# of obs.**	**Avg. (min)**	**Std.Dev. (min)**	
1	C789	6	1.29	1.47	1.11
1	T220	5	1.69	1.70	1.57
1	T240	10	1.12	1.10	0.22
2	C785	3	1.67	1.84	1.57
2	CT220	10	2.32	2.55	0.11

Table 3-10 Comparison of Actual and Predicted System Productions for Scenario 2

Excavator	**Ideal Production (LCM/Hr)**	**Field Observation**	**AP2-Earth Simulation Results**	**Comparison**	
		Production (LCM/Hr)	**Production (LCM/Hr)**	**Diff.**	**Percent Error**
EX-3500	1214	1190	1117	-73	-6.1%
ONK	1119	840	923	83	9.9%

3.5.7 Summary of Study

The case study presented shows that AP2-Earth is able to predict, with reasonable accuracy, numerous important parameters of large earth-moving projects. The predicted truck travel times, which are based on manufacturer specifications and simulated interaction delays, differed by less than 8% on average. The waiting duration in the loading area was predicted with reasonable accuracy. The predicted system production, which is the most important parameter because it includes all simulation errors and assumptions, was accurate to an average of 6.5%.

3.6 Conclusion

A specialty construction simulation tool has been presented for the analysis of earth-moving projects. The success of the developed system was due to its accuracy, scalability and integrated design. As well, the graphical user interface and the high level modeling constructs allowed users who were not familiar with simulation theory to build sophisticated models and experiment with various road layouts and resource allocations in order to optimize the overall production.

Chapter 4 – Aggregate Production Plant Simulation[1]

4.1 Introduction

This chapter presents the main design and implementation steps of a specialized simulation tool called CRUISER. The system was developed in cooperation with a local aggregate production contractor. Similarly to the AP2-Earth development, one objective of the system was to introduce the aggregate industry to a simulation based analysis for optimizing aggregate plant operations. The other objective was to gather information on the general features that a simulation tool for the aggregate production industry must possess in order to be accepted and successful.

General background related to aggregate production is first provided in section 4.2 followed by a detailed description of the computer simulation tool that can be used to model and simulate a full crushing plant in section 4.3 . Section 4.4 presents the results of a case study performed to validate the developed model, followed by the conclusions in section 4.5 .

4.2 Aggregate Production

Crushed-stone aggregate production involves material size reduction and separation of raw or crudely broken material into a form that is suitable for use in construction. The objective is to obtain aggregates that satisfy the quality requirements of the end user at the minimum possible cost. Decisions regarding plant location, configuration and equipment settings are typically based on the experience of field personnel.

---

[1] A version of this chapter was published as "Modeling and Analysis of Aggregate Production Operations" in the Journal of Construction Engineering and Management, ASCE, Volume 124, Number 5, September 1998.

The production process of construction aggregates involves size reduction and separation as well as many supporting operations such as drilling, blasting, loading, transporting, and product handling. The methods discussed in this study apply only to the processing phase. For a general overview of the theory and practice associated with these activities, the reader can refer to (Peurifoy, Ledbetter, and Schexnayder 1996), and (Nunnaly, 1980).

4.2.1 Aggregate Processing

Aggregate processing consists of two main activities: material size reduction and size separation. Size reduction is achieved through the use of specialized crushing equipment which operate on the principles of nipping (jaw, gyratory and cone crushers) or high impact (single and double impeller impactors) (Smith and Collis 1993). Most aggregate sizing methods are carried out by means of a screening operation, where particles are graded according to their minimum cross-sectional length. Typical industrial screens support multiple decks with varying opening sizes. In simple static screens, transport is arranged by inclining the surface sufficiently for material to be moved by gravity. Other types of screens utilize a combination of gravity and rotating motion that is supplied by applying vibration (Peurifoy, Ledbetter, and Schexnayder 1996).

4.2.2 Multi-Stage and Cyclic Processing

Aggregate production plants utilize several stages for processing where material is passed through several crushers and screens as demonstrated in Figure 4-1. This procedure is used as a means of controlling the quality of the finished product. The amount of size reduction accomplished is directly related to the amount of energy used. Higher energy outputs lead to higher reduction ratios but can also lead to higher percentage of fines. Fines are mostly non-revenue by-products that are too small to be used as aggregates. Therefore, using several crushers can help to minimize waste by distributing the required energy. Typical plants utilize two to three crushers along with several screens. Conveyers are the primary means of material transport between the equipment. Material entering the plant follows a given path along conveyors. This path will be referred to as a "stream" in this chapter.

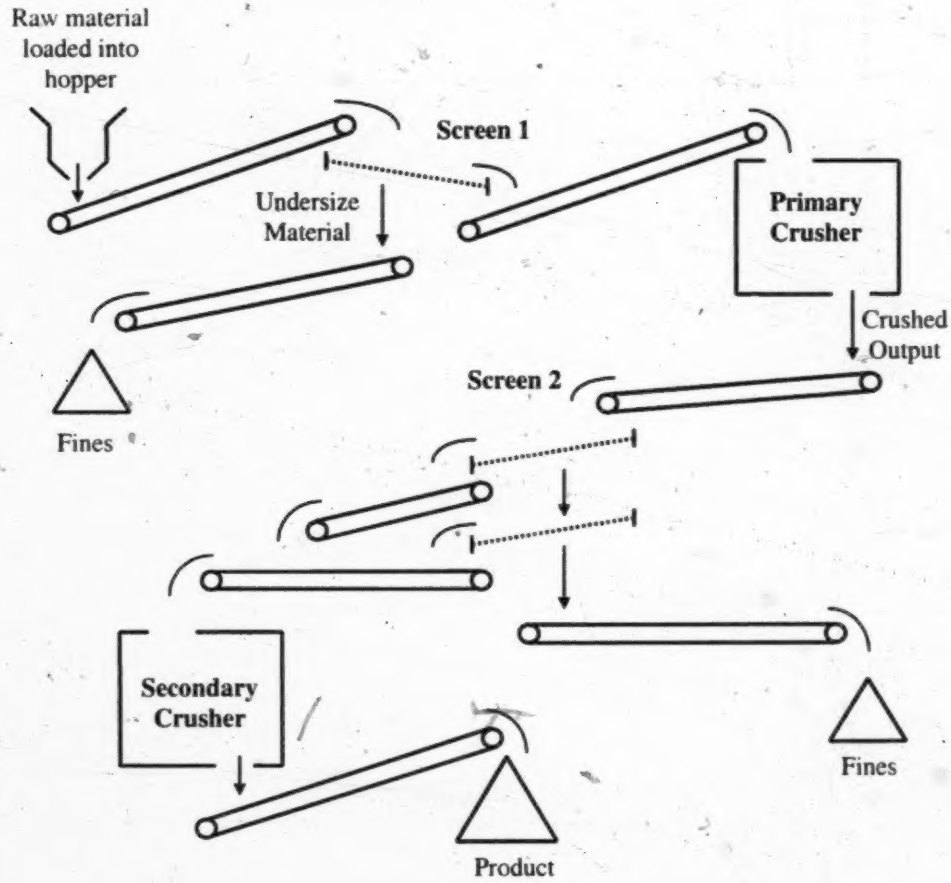

Raw material loaded into hopper
Screen 1
Undersize Material
Primary Crusher
Crushed Output
Fines
Screen 2
Secondary Crusher
Fines
Product

Figure 4-1 Typical Aggregate Plant Layout

Aggregate processing operations are commonly divided into two classifications: "open circuit" and "closed circuit". In an open circuit operation, aggregate streams flow through each component once as they make their way to the stockpile. In a closed system operation, at least one stream is cycled back to a previous stage of crushing as in Figure 4-2. Closed circuit operations require relatively uniform feed gradation and tonnage as well as compatible settings between the crusher and the controlling screen. If any of these conditions fail, then material will gradually build up until the equipment is overloaded. In a carefully managed operation, the circulating load will gradually build at plant startup until a steady state level is reached.

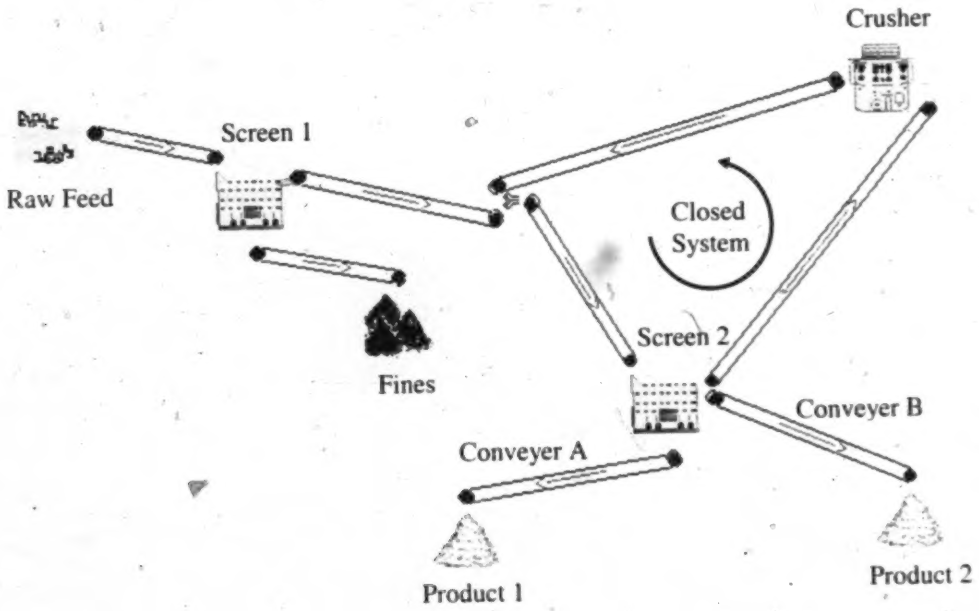

Crusher
Screen 1
Raw Feed
Closed System
Fines
Screen 2
Conveyer B
Conveyer A
Product 1
Product 2

Figure 4-2 Example of Closed System Operations

4.2.3 Quality

Aggregates are classified according to several aspects of their physical and chemical properties. Such properties include strength, moisture content, percent fractures, clay content, organic material content and product gradation. Quality is defined as conformance of these properties to end user specifications. Quality control during aggregate processing mostly involves the control of product gradation obtained using sieve analysis. Results are normally presented graphically as percent passing or cumulative percent retained on standard sieve sizes (ASTM 1996). Specifications are provided as upper and lower bounds on the gradation in the form of an envelope. A sample test result that conforms to specifications is shown in Figure 4-3.

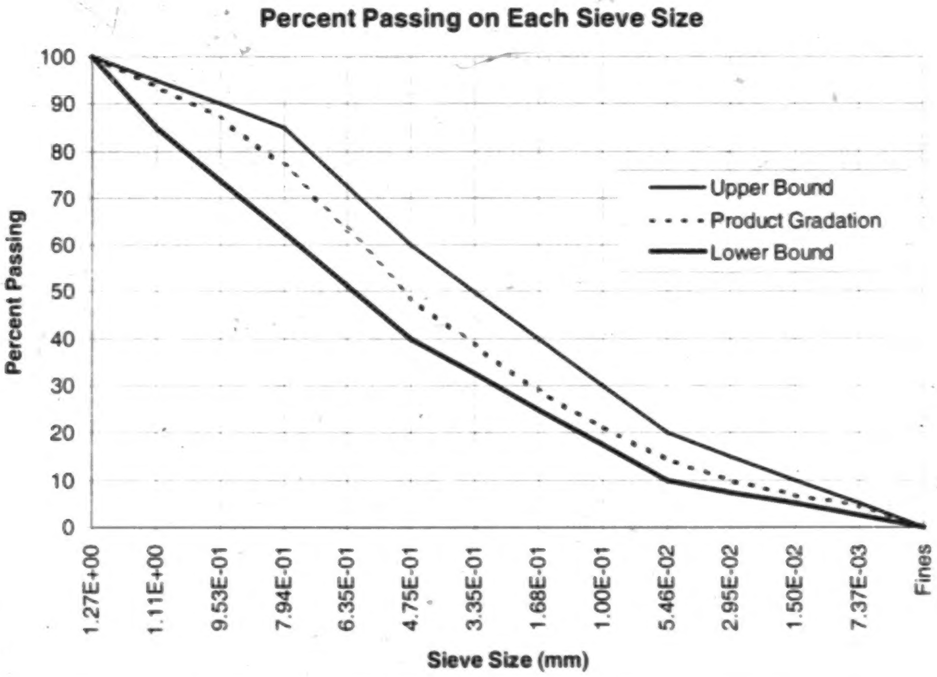


Figure 4-3 Sample Sieve Analysis Results

4.2.4 Revenue

Revenue from aggregate production is affected by decisions at two levels. At the regional or corporate level, managers are concerned with the allocation of available equipment across a number of potential pits. Preliminary geological surveys determine the type and volume of gravel sediments. This information, along with historical production estimates is used to determine the plant locations. At the plant level, the objective is normally to maximize production. This can generally be achieved by optimizing the crusher settings and screen sizes. Experienced field managers can make these decisions through a visual inspection of the plant along with an examination of the product gradation.

4.3 Simulation of Aggregate Processing

4.3.1 Overview

Computer simulation of the aggregate production process can greatly improve operations at any level. Corporate decision-makers can use the tool to obtain accurate estimates of expected productions before any resources are committed. At the site level, simulation allows field managers to prototype their decisions before implementing them. They would have the ability to experiment with various scenarios (for example, a different screen size or crusher setting) and then observe the simulated effect on quality and production. Such analysis does not disrupt plant production, which reduces lost revenue associated with plant shutdowns.

A simulation tool for aggregate production called CRUISER was developed. The objective was to create a highly visual and user friendly environment for the definition and manipulation of the model. The main step in the development process consists of the definition of classes for the representation and manipulation of the various processes (eg. raw feeds, conveyors, screens, and crushers). The developed class hierarchy is shown in Figure 4-4. Table 4-1 provides a brief overview of the abstract classes.

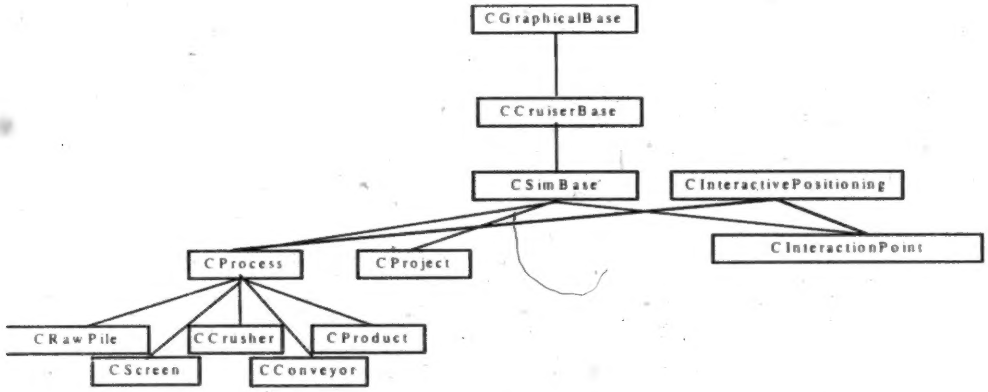


Figure 4-4 CRUISER Class Hierarchy

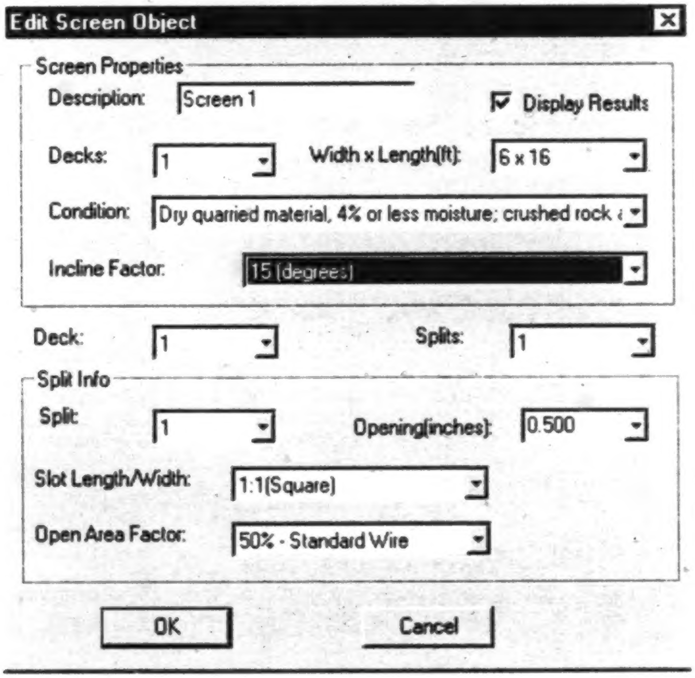
Table 4-1 CRUISER Abstract Classes

Class	Functionality	Comment
CGraphicalBase	Basic graphical state and position representation/ manipulation functionality.	This class aids in satisfying the objective of graphical manipulation by allowing objects to posses visual editing capabilities.
CInteractive-Positioning	"click and drag" interactive positioning functionality	This added behavior simplifies data entry for position and linking information of all classes.
CSimBase	Simulation behavior support	This class provides various "user hooks" that allow the specialized classes to perform custom manipulation of numerous behaviors.
CProcess	Basic Process Definition	A CProcess class defines basic functionality of a high level process such as ability to connect to interaction points and the base structures needed to define a simulation model. Child classes (i.e. CCrusher, CScreen,...) implement the specific simulation models.
Cproject	Project level information and resources	Project level information includes the list of all processes, interaction point definitions, as well as any global resources that are available to all the processes (i.e. standard gradation envelops,...) .

In general, a CRUISER object (i.e. an instance of a non-abstract class, typically at the lowest level of the hierarchy) encapsulates a variety of information and methods depending on the inheritance branch. The supported functionality provided by the abstract classes includes: graphical behavior, simulation behavior, and reporting behavior.

4.3.2 Graphical Behavior

Graphical representation and manipulation functionality is gained by deriving from the CGraphicalBase class. Classes deriving from this base class are required to implement certain virtual functions in order to conform to the overall visual manipulation objective. Each class typically has an associated graphical dialog box where the user can enter and change information in a very intuitive manner. Figure 4-5 shows the CScreen dialog box, which allows users to specify screen specific attributes. Figure 4-6 shows the CConveyor dialog box where stream properties such as gradation and production rate are displayed. The graphical behavior of the CProject class is, in fact, the main program screen where CRUISER objects can be added, deleted, or linked. This screen is shown in Figure 4-7.



Description: Screen 1

☑ Display Results

Decks: 1

Width x Length(ft): 6 x 16

Condition: Dry quarried material, 4% or less moisture; crushed rock

Incline Factor: 15 (degrees)

Deck: 1

Splits: 1

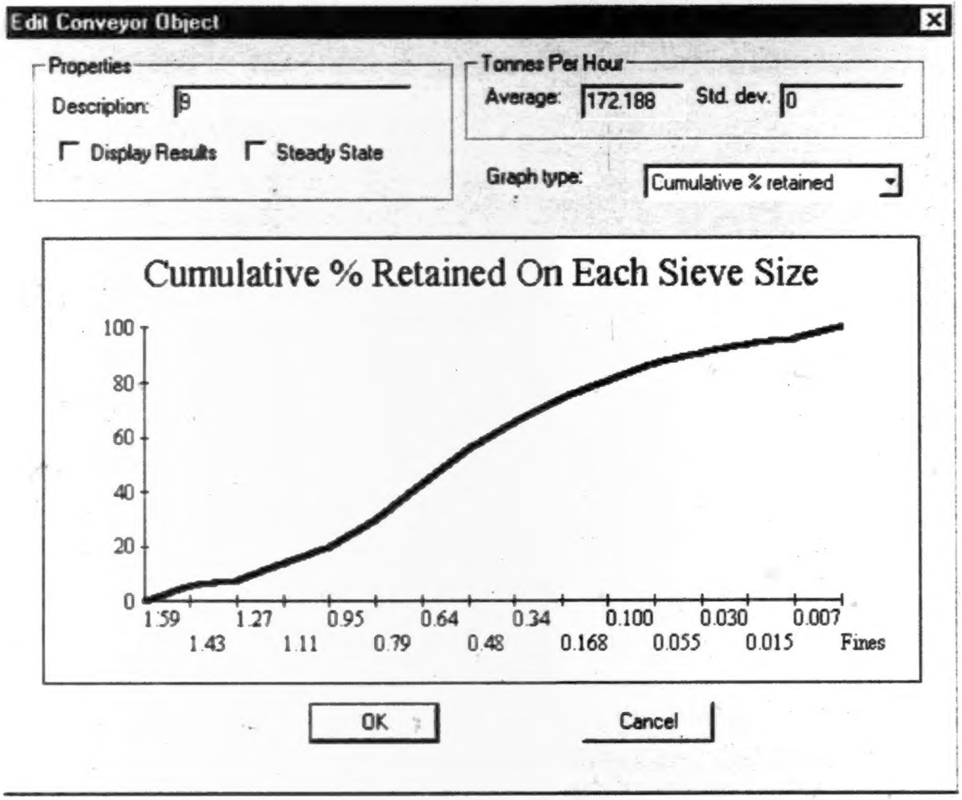
Split: 1

Opening(inches): 0.500

Slot Length/Width: 1:1(Square)

Open Area Factor: 50% - Standard Wire

Figure 4-5 Screen Dialog Box


Edit Conveyor Object
Properties
Tonnes Per Hour
Cumulative % Retained On Each Sieve Size
100
80
60
40
20
0
1.59
1.43
1.27
1.11
0.95
0.79
0.64
0.48
0.34
0.168
0.100
0.055
0.030
0.015
0.007
Fines
OK
Cancel

Description: 9

☐ Display Results ☐ Steady State

Average: 172.188 Std. dev. 0

Graph type: Cumulative % retained

Figure 4-6 Conveyer Dialog Box

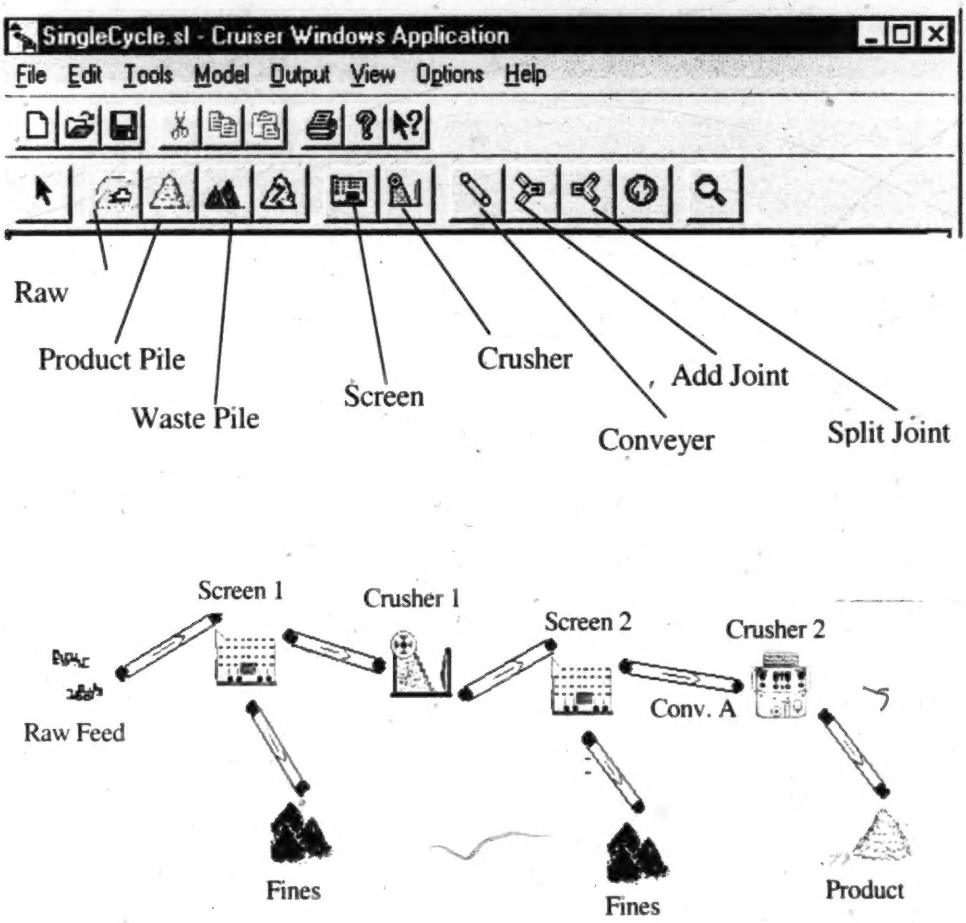

SingleCycle.sl - Cruiser Windows Application
File
Edit
Tools
Model
Output
View
Options
Help
Raw
Product Pile
Waste Pile
Screen
Crusher
Add Joint
Conveyer
Split Joint
Screen 1
Crusher 1
Screen 2
Crusher 2
Raw Feed
Conv. A
Fines
Fines
Product

Figure 4-7 Main Modeling Components

4.3.3 Simulation Behavior

Simulation tools typically implement the simulation code in a single module. This meant that minor modifications could potentially require changes to a large number of functions spread across many files. With an object-oriented modeling approach however, the simulation model representations are separated and implemented by the individual classes. In this context, each class derived from CSimBase defines a unique simulation model and becomes an independent entity in every respect. In CRUISER, the individual simulation models perform a variety of functions including stream initialization, stream transformation and other required analysis. The following sections will detail the simulation models used for each class.

4.3.4 CRawPile

The CRawPile Class models the properties of the input stream that represents information collected by sampling the gravel pit material. The testing process is usually subject to many variations and, as a result, stochastic simulation is used in its modeling. For such simulation, randomness is usually driven by random input processes. For gradation, a normal distribution is used to model the amount retained on each sieve size, while the production rate is modeled as a triangular distribution. The implication of this type of modeling approach is that the analysis is carried as a statistical experiment where the simulation is repeated a number of times, output measures are collected from each simulation run, and point estimates of various statistics along with their confidence intervals are derived for use in decision making. When the CRawPile's simulation function is called, a new random stream based on the user's parameters is generated and used as input to components that follows.

4.3.5 CProduct

The simulation function of the CProduct class is used to perform statistical analysis after the completion of all simulation runs. For gradation, the mean values of desired parameters are calculated and a 95% confidence interval is constructed for each. The three sets of results (low, mean, high) are displayed graphically as percent passing or cumulative percent retained. For all other stream properties including hourly production, the average and standard deviation are provided.

4.3.6 CConveyor

The CConveyor class holds information ~~regarding the~~ carried stream as well as generic conveyer properties such as elevation and length. Stream information stores the intermediate stream properties from all runs to be analyzed once simulation is completed. The other properties help determine the conveyer horsepower, width, type and speed that are required to transport the predicted load based on published equipment manufacturer charts (e.g. Pioneer 1988 or Cederapids 1994a). The CConveyor class does not have a simulation component as it is assumed that the transported stream properties are continuous and will not change.

4.3.7 CCrusher

The CCrusher simulation function models the size reduction process. The size reduction or crushing process is highly complex and little theory is available to allow for its accurate modeling. As a result, studies have been based on modeling the crushing process using empirical data in the form of performance tables like the one shown in Figure 4-8.

Setting	Size (mm)									
(mm)	25.4	22.9	20.3	17.8	...	0.05	0.04	0.03	0.01	0
1.905	100	100	100	100	...	3.8	2.9	2.3	1.3	0
2.54	100	100	100	100	...	2.9	2.1	1.6	1	0
3.175	100	100	100	100	...	2.5	1.9	1.4	0.8	0
3.81	100	100	100	100	...	2.2	1.7	1.3	0.7	0
...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...
12.7	100	100	93	83	...	0.9	0.7	0.6	0.3	0
15.24	94	88	80	72	...	0.8	0.6	0.5	0.3	0
17.78	82	76	69	62	...	0.6	0.5	0.4	0.2	0
20.32	73	68	61	54	...	0.6	0.5	0.4	0.2	0
22.86	66	61	55	48.5	...	0.4	0.3	0.2	0.1	0
25.4	60	55	50	44	...	0.3	0.24	0.2	0.1	0

Figure 4-8 Sample Crusher Performance Chart

The table gives the expected product gradation based on a certain crusher setting. The data provided in these tables represents an average output across a range of feed rates and material types. To be used accurately, the basic predicted gradation must be adjusted for the properties of the input gradation. The following is a summary of the algorithm which is based on a method suggested by Hancher and Havers (1972):

1. If there is no oversize (material larger than crusher setting) in the input stream then the output stream is the same as the input stream. This is a simplifying assumption because in general, a small percentage of material might in fact undergo some breakage as it jams in the crusher.
2. If input stream load exceeds crusher capacity then halt analysis and give warning.
3. Calculate effective tonnage to crush. Effective tonnage is the sum of materials larger than half the crusher setting and smaller than one and a half the crusher setting. The assumption is that all material above 1.5 crusher setting will be crushed and all material less than 0.5 crusher setting will not be affected.

4. Set initial output gradation to expected gradation as given by empirical tables.
5. If amount of oversize in output exceeds that of the input then adjust oversize in output.
6. Add material that is unaffected by the crushing process (less than 0.5 crusher setting) to the output gradation.

4.3.8 CScreen

The CScreen simulation function models the size separation process. There are several issues to consider when modeling the operations of a screen. Size separation is never a perfect operation and a certain quantity of undersize material will generally end up in the oversize stream. Further, multi-deck screens require that each deck be analyzed separately where output streams from top decks are used as inputs into the lower levels. In certain cases, managers use different size screen meshes or splits on the same deck. For example, the first half of the deck has a ten millimeter opening size while the next half utilizes a five millimeter size. This technique is used to distribute the load across two decks and to overcome the limited capacity of certain screens.

Screening efficiency is a measure of the effectiveness of a screen in separating an incoming stream into oversize and undersize streams. For example, an 80% value for efficiency indicates that 20% of undersize material fails to pass through the openings and ends up in the oversize stream. Typical values range between 60 and 95 percent. Efficiency of a given screen can be predicted using the Allis-Chalmer relationship shown in Figure 4-9 (Hancher and Havers, 1972). Efficiency is assumed to be directly dependent on the ratio of actual load to rated capacity. Rated capacity consists of the basic capacity, which is only dependent on opening size and surface area, adjusted for several factors including screen incline angle, moisture content, and other properties of the incoming stream. For a complete overview of all the factors and their effect on screen capacities see Cederapids 1994b.

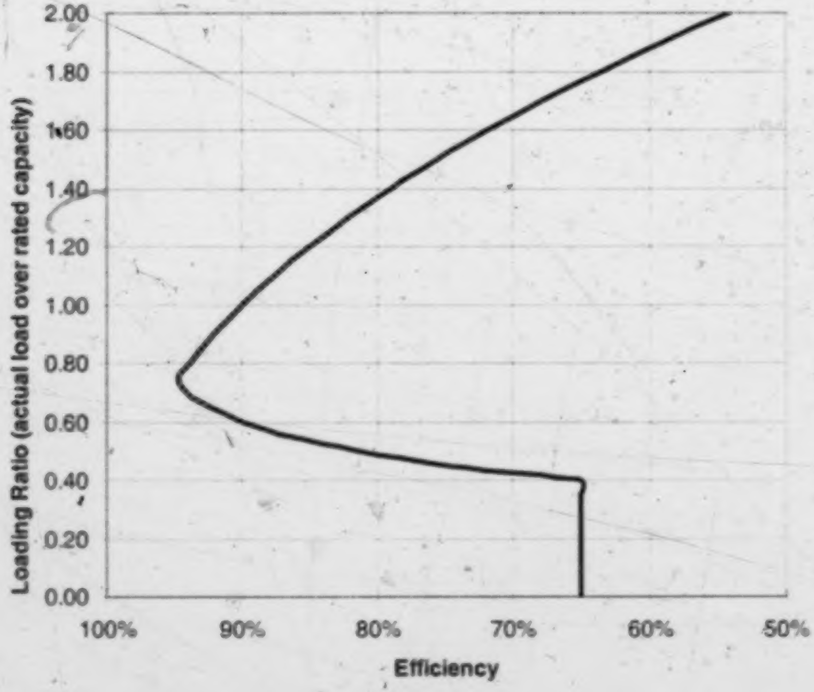


Figure 4-9 Allis-Chalmer Relationship For Calculating Screening Efficiency

Once efficiency is calculated, the size distribution of undersize particles that did not pass through is determined. From experience, if the screen is operating above the optimum loading ratio, the undersize material contained in the oversize stream is equally distributed. As the loading ratio decreases below the optimum point, the percentage of larger sizes increases exponentially. The authors have modeled this relationship as is illustrated in Figure 4-10. The overall size separation algorithm is summarized as follows:

1. Calculate rated capacity.

2. Calculate loading ratio using rated capacity and incoming stream load.
3. Calculate efficiency using Allis-Chalmer relationship.
4. Calculate undersize and oversize streams based on 100% efficiency.
5. Adjust for efficiency by removing appropriate amount from undersize stream and adding to oversize stream. If loading ratio is greater than 75%, then size distribution of undersize material is equally distributed, otherwise distribution is determined using the relationship shown in Figure 4-10.

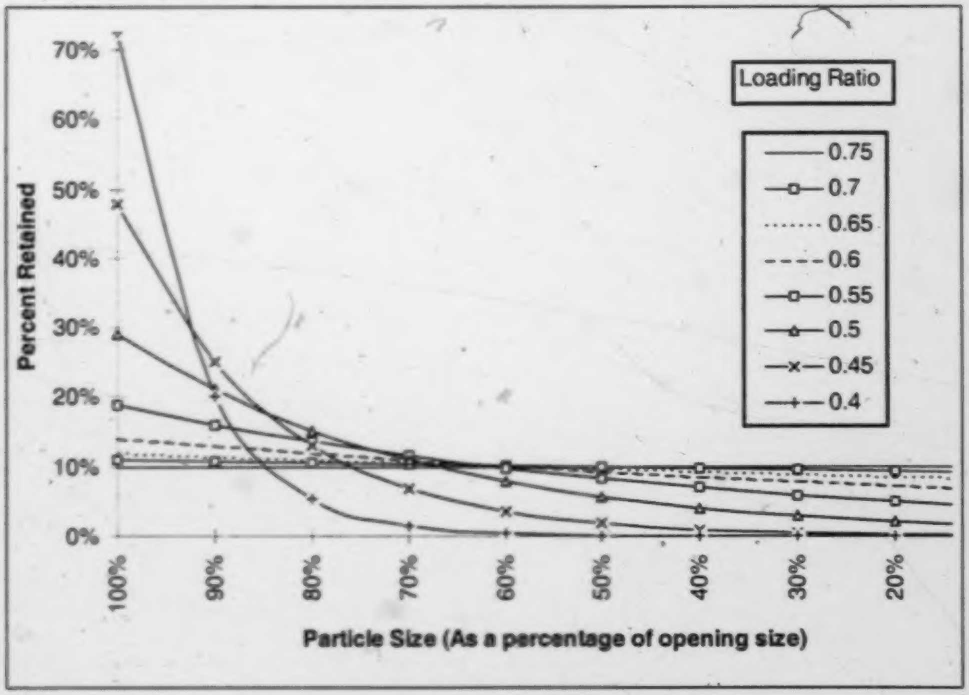


Figure 4-10 Distribution of Undersize Particles Based on Loading Ratio

For multiple deck and multiple split screens, a general procedure that analyzes each split as a separate screen was developed as shown Figure 4-11. Essentially, the incoming stream is distributed between the first two splits with the majority directed towards "split 1". The oversize stream from "split 1" is combined with incoming stream as input to "split 2" and so on for the subsequent splits. The undersize streams from the top deck are combined and distributed across the second deck splits based on the relative surface area of each. This is a simplifying assumption since geometry and the relative position of the splits could affect the distribution. The analysis is similar for the other decks. Finally, undersize from the bottom deck is combined to form the undersize stream of the entire screen.

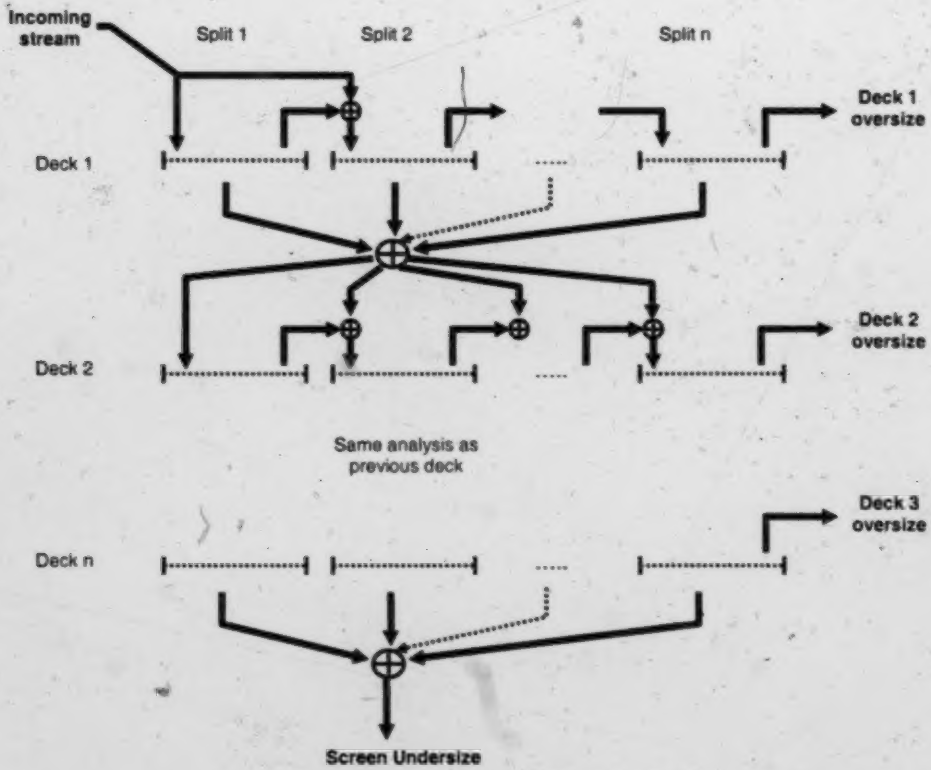

Incoming stream
Split 1
Split 2
Split n
Deck 1 oversize
Deck 1
Deck 2 oversize
Deck 2
Same analysis as previous deck
Deck 3 oversize
Deck n
Screen Undersize

Figure 4-11 Multi-Deck Multi-Split Screen Analysis Procedure

4.3.9 CProject

The simulation function of the CProject class is the driving mechanism of the entire simulation process. The modeling of an aggregate production plant involves the analysis of the material flow as it makes its way through the processing equipment in the appropriate order. The starting point is normally a "raw pile" where a loader generally feeds a hopper, which provides feed at a relatively uniform rate to the plant. The analysis proceeds by following this stream as it makes its way through the plant. The initial stream will eventually split at various plant locations, creating other streams. A split generally occurs when a screen is encountered. When this occurs, the analysis continues along both branches in parallel. Analysis along any branch is completed only when a product or waste stockpile is reached. The entire analysis stops when all streams have been analyzed.

The CProject's simulation function utilizes an event-based simulation engine developed for aggregate processing. The basic event scheduling mechanism is used to perform a structured and orderly analysis of any plant layout. At the heart of an event driven algorithm is the event queue. It determines the next event or task, which must be performed. Each scheduled event holds information pertaining to the type and identity of the equipment being modeled. The algorithm for the simulation engine works by executing the following tasks continually until the event queue is empty:

1. obtain the next queued event
2. cross-reference the object information using the event properties
3. call the object's simulation function
4. schedule future events.

The number of scheduled events resulting from the fourth step will vary depending on the current component being analyzed. Conveyers, crushers and raw feed objects will cause a single event corresponding to the next connected equipment to be scheduled. Screens will cause one event to be scheduled for each deck oversize as well as for the undersize stream.

To see how this algorithm operates, consider the scenario shown in Figure 4-12. The contents of the event queue at each simulation step are shown in Table 4-2. The queue is initialized with Event 1, corresponding to the raw feed object, at the start of simulation. The simulation proceeds by following the streams that flow through the various objects. The simulation is terminated when the event queue is empty.

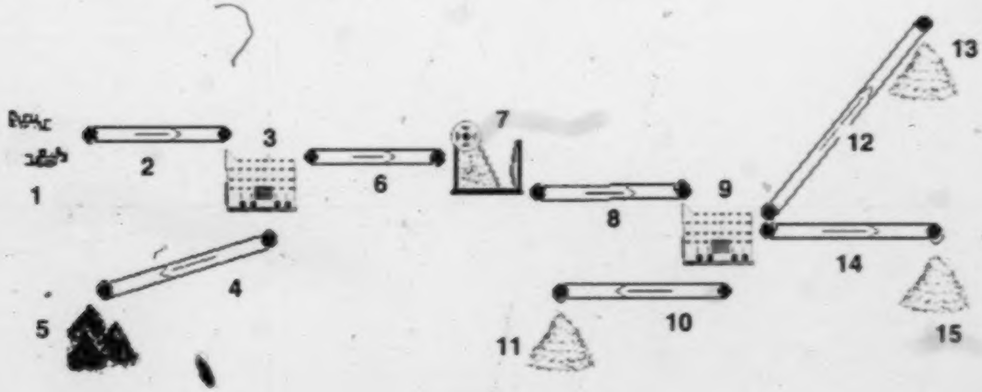


Figure 4-12 Example Plant Flow Simulation

Closed cycle systems require special analysis consideration as they must be analyzed continually until a steady state production is reached at the exit conveyer. The exit conveyer is that which completes the cycling loop (conveyers A and B in Figure 4-2). This task is automated by recording the calculated production rate for each cycle analysis.

Successive observations are recorded across an interval and steady state is assumed when the difference between the maximum and minimum values reaches a specified threshold.

Table 4-2 Contents of Event Queue

Simulation Time	Event queue Content	Notes
0	1	Starting simulation with event 1
1	2	Conveyer 2 analysis is scheduled
2	3	Screen 3 analysis is scheduled
3	4,6	Conveyer 4 and 6 analysis scheduled
4	5,7	Waste pile 5 and crusher 7 analysis scheduled
5	8	Conveyer 8 analysis schedule
6	9	Screen 9 analysis scheduled
7	10,12,14	Conveyer 10,12,14 analysis scheduled
8	11,13,15	Product piles 11,13,15 analysis scheduled
9	Empty	Event queue empty so simulation is terminated

4.3.10 Reporting Behavior

Each CRUISER class is able to take advantage of reporting functions designed to collect the results of the simulation and present the information to the user. An example of such a function is the OnReportValidity(), which is used to report integrity errors such as missing connections between components. This information helps the user to locate the sources of modeling errors so that they can be quickly rectified.

4.3.11 CRUISER Environment

The steps used to analyze a proposed plant are illustrated in Figure 4-13. The user defines the model by selecting from a list of provided modeling objects, which represent various types of crushers, screens, stockpiles and conveyers as previously shown in Figure 4-7. The properties of each object can be manipulated through the component dialog box, which can be accessed by double clicking on its graphical icon. The dialog box for the "RawPile" object is illustrated in Figure 4-14. The model can be interactively updated at any time. All objects can be moved around the screen by clicking and dragging. This allows the user to create models that greatly resemble the real life situation.

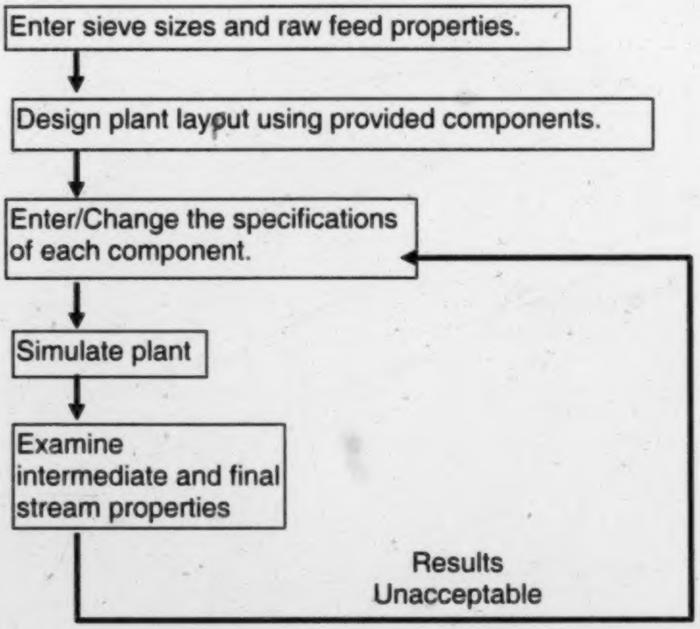
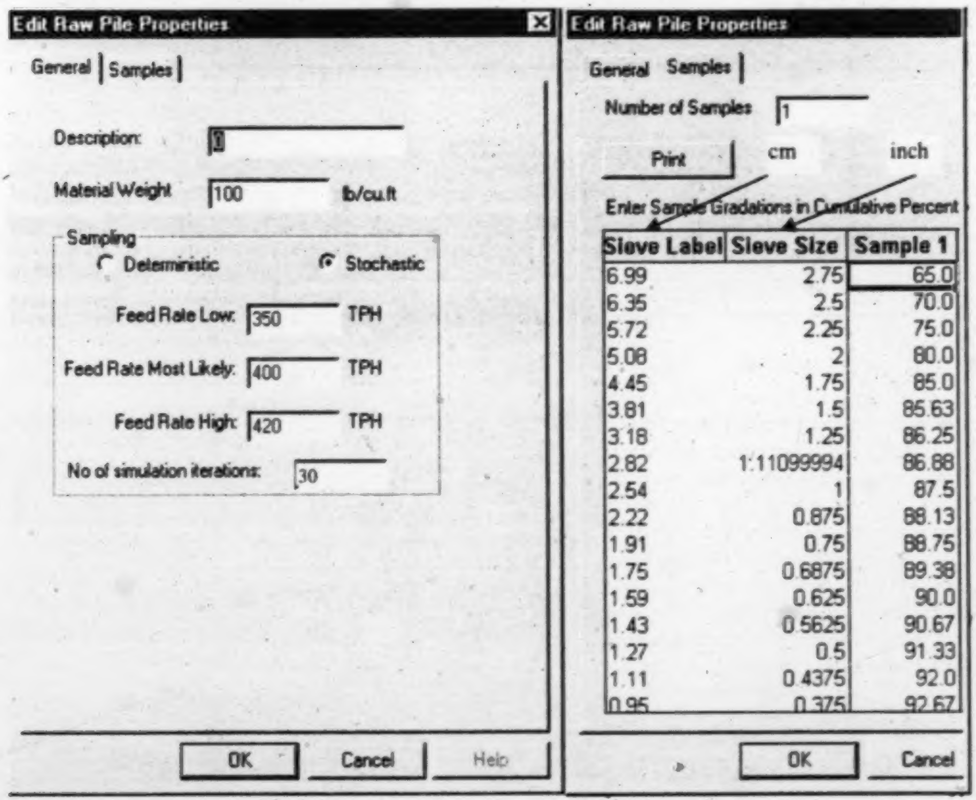

Enter sieve sizes and raw feed properties.
Design plant layout using provided components.
Enter/Change the specifications of each component.
Simulate plant
Examine intermediate and final stream properties
Results Unacceptable

Figure 4-13 Plant Design Steps


Edit Raw Pile Properties
General
Samples
OK
Cancel
Help
Edit Raw Pile Properties
General
Samples
Print
cm
inch
OK
Cancel

Description:

Material Weight: 100 lb/cu.ft

Sampling

○ Deterministic ● Stochastic

Feed Rate Low: 350 TPH

Feed Rate Most Likely: 400 TPH

Feed Rate High: 420 TPH

No of simulation iterations: 30

Number of Samples: 1

Enter Sample Gradations in Cumulative Percent

Sieve Label	Sieve Size	Sample 1
6.99	2.75	65.0
6.35	2.5	70.0
5.72	2.25	75.0
5.08	2	80.0
4.45	1.75	85.0
3.81	1.5	85.63
3.18	1.25	86.25
2.82	1.11099994	86.88
2.54	1	87.5
2.22	0.875	88.13
1.91	0.75	88.75
1.75	0.6875	89.38
1.59	0.625	90.0
1.43	0.5625	90.67
1.27	0.5	91.33
1.11	0.4375	92.0
0.95	0.375	92.67

Figure 4-14 Raw Input Stream Modeling

4.4 Model Validation

The developed system was implemented at a number of gravel pits for field-testing. One of the case studies performed involved a crusher plant located in Edmonton, Canada. The pit run gravel in the area was small to medium size, requiring two cone crushers, one jaw crusher, and three double-deck screens. The model for this plant is shown in Figure 4-15. The actual conditions observed indicated that the raw feed material was supplied at an average rate of 520 tons/hour and the actual output production was equal to an average of

250 tons/hour. The rest of the material is considered very fine. The average input raw feed gradation is shown in Table 4-3.

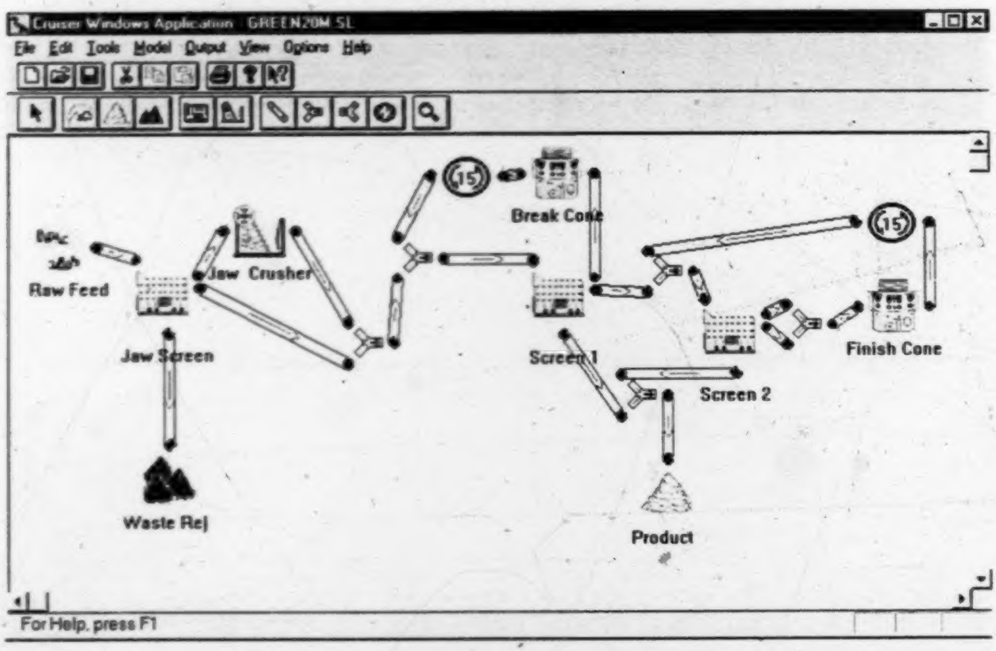

Cruiser Windows Application - GREEN20M.SL
File Edit Tools Model Output View Options Help
Raw Feed
Jaw Screen
Jaw Crusher
Waste Rej
15
Break Cone
15
Screen 1
Screen 2
Finish Cone
Product
For Help, press F1

Figure 4-15 Plant Layout for Case Study

Table 4-3 Raw Feed Gradation

Sieve mm	Cumulative % Retained
63	21.47%
50	21.47%
40	27.90%
28	27.90%
25	48.02%
20	54.73%
16	54.73%
14	54.73%
12.5	64.11%
10	68.40%
5	74.49%
2.5	74.49%
2	77.36%
1.25	77.36%

0.8	79.46%
0.63	79.46%
0.4	85.99%
0.315	85.99%
0.16	96.49%
0.08	99.35%
pan	100.00%

The "Jaw Screen" is a 1.64 m x 5.9 m double deck inclined vibrating screen with a top deck opening size of 5.5 cm, and a second deck opening size of 2.2 cm, and a slope of 15 degrees. "Screen 1" is a 1.64 m x 5.25 m double deck vibrating screen with a top deck opening size of 4.78 cm and a second deck opening size of 2.05 cm. "Screen 2" is a 1.64 m x 5.25 m double deck vibrating screen with a top deck opening size of 3.08 cm and a second deck opening size of 2.39 cm.

The jaw crusher has a clear opening distance (setting) of 6.8 cm and an estimated capacity of 95 tons/hr. The breaker cone has a clear opening distance of 2.7 cm and an estimated capacity of 181 tons/hr. The finish cone has a clear opening distance of 2.05 cm and an estimated capacity of 154 tons/hr.

The results predicted by CRUISER are compared to the actual sieve analysis of the product pile as shown in Figure 4-16. The predicted production is 236.94 tons/hr as compared to the 250 tons/hr observed on site. A summary of two other case studies performed is shown in Figure 4-17. The results showed average errors of 12.68% and 5.37%, respectively (Chehayeb 1997). These results are consistent with original findings by Hancher and Haver (1972). Some of the deviations could be attributed to sampling errors, others reflect the generality of the underlying crushing tables when applied to specific crushers.

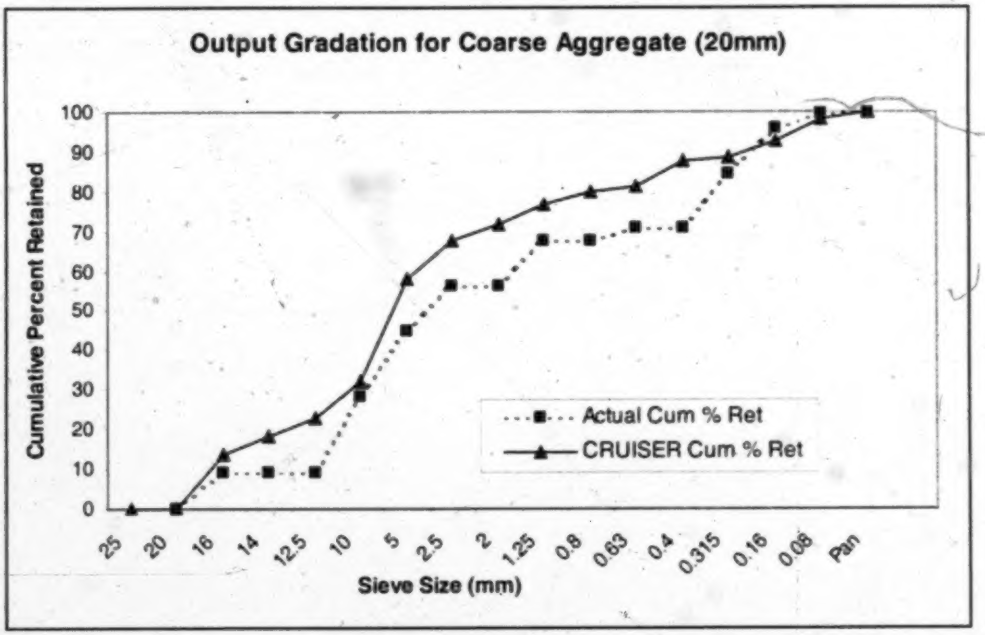
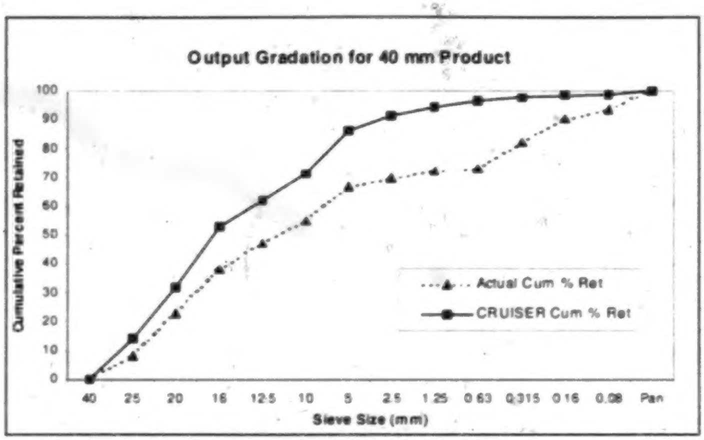
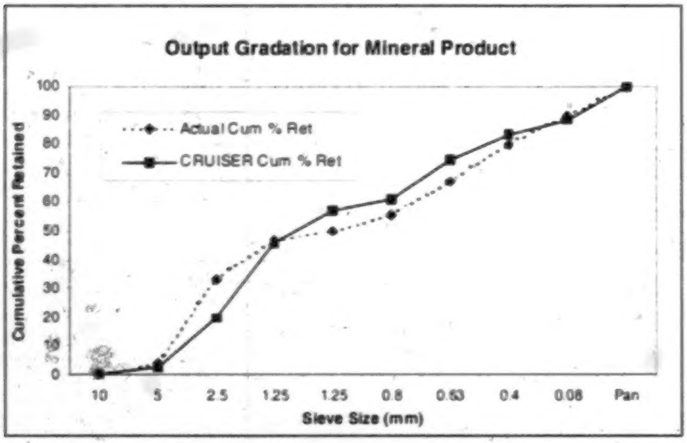


Figure 4-16 Comparison of CRUISER Predicted Output Vs Actual Gradations



Case 2: Kapasawin Crushing Plant



Case 3: Villneuve Crushing Plant

Figure 4-17 CRUISER Case Study Results

4.5 Conclusion

This chapter discussed the successful development and implementation of a tool for the computer simulation of aggregate production plants. CRUISER was successfully validated and implemented with several local aggregate producers. This success can be attributed to many factors. First, the graphical modeling environment gives the tool a user-friendly interface for the definition and manipulation of complex plants. Objects can be created and manipulated using simple "click and drag" mouse operations. Users do not need to posses any simulation knowledge in order to use the tool. Second, the simulation modeling constructs are extremely flexible and are capable of representing a large number of crushing plant scenarios. Third, due to the independence of CRUISER objects and the defined connection mechanisms, the tool is highly scaleable and can be easily extended since the addition of new modules does not require any changes to be done to the existing classes. For example, users can modify the general empirical tables that are used to predict output gradation in order to model specific equipment or rock types. Researchers who are not content with using empirical tables for performing the analysis can easily substitute alternative predictive models such as neural networks.

Chapter 5 – Construction Site Dewatering Simulation[1]

5.1 Introduction

This chapter presents a computer tool for the modeling and analysis of construction site dewatering operations called CSD. The tool allows for site modeling in a manner that is intuitive, user-friendly and natural for use by construction engineers. The fundamentals of site dewatering analysis and the tool design and implementation steps are discussed. CSD was developed in cooperation with a local general contractor. The described research resulted in the development of a tool that was of immediate use for the collaborating construction company and allowed for the identification of key features that such tools must possess.

This chapter is organized as follows: Section 5.2 provides some background on the construction site dewatering process. Section 5.3 details the structure and operation of CSD. Section 5.4 presents a case study used to validate the accuracy and relevance of the underlying model. The conclusions are then presented in Section 5.5 .

5.2 Background

Among the more common problems in construction work is the need to handle subsurface water encountered during and after construction. During construction, removal of water from working areas is desirable to provide both workers and equipment with better working conditions. Construction dewatering is not an easy task to achieve, especially when excavations extend more than a few feet below groundwater. In this case, open ditches are not practical and well-point systems or deep wells are normally used. One of the main problems associated with the use of deep wells and well-point systems for construction dewatering is that of defining the best possible well configurations (i.e. well positions and pumping rates) that result in the least pumping effort and hence, the lowest dewatering costs. Such decisions are often made by site engineers based on experience and hydrological analysis. Numerous hydrological models exist for this type of analysis, including steady state models based on equations developed by Muskat (1953) and Thiem (1906) or finite-difference based methods such as those used by MODFLOW (McDonald and Harbaugh 1988). The type of hydrological model used generally depends on the amount and nature of the available information. Steady state models make simplifying assumptions about the uniformity and homogeneity of the underground soil layers. For situations requiring an analysis based on relatively non-standard elements such as complex recharge areas, ponds, and non-homogenous layers, more complex models such as MODFLOW are used.

---

[1] A version of this chapter was published as "Optimizing Construction Site Dewatering Operations using CSD" in the Canadian Journal of Civil Engineering, CSCE, Volume 25, Number 3, June 1998.

Construction site dewatering projects can obviously benefit from the use of such models. Unfortunately, the writers' experience with local construction firms indicates that contractors do not complete the required dewatering analysis for most of their jobs. Construction engineers generally lack the knowledge required to complete a dewatering study. Furthermore, they do not possess the proficiency required to use tools such as MODFLOW. Contractors, however, can lose money on a project if the dewatering analysis is not correctly done. As a minimum, it is desirable to complete an appropriate analysis to validate dewatering subcontractor estimates. The above provided the motivation for the work described in this chapter.

5.3 System Description

Figure 5-1 illustrates the developed components and the application's function within an integrated construction information system. The external modules shown are provided as an example of the potential capability of CSD to integrate with other systems. The following subsections will provide details about the structure of each module.

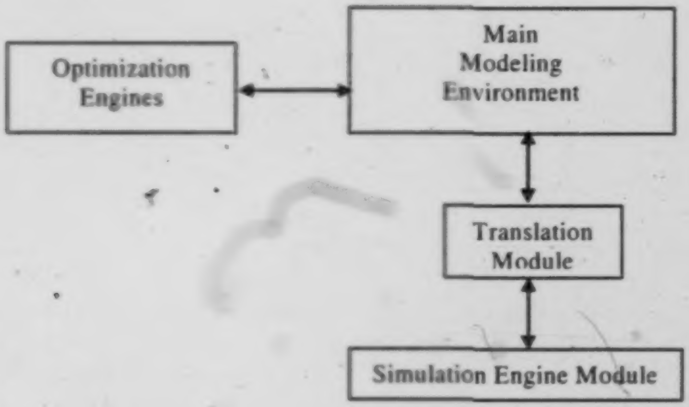


Figure 5-1 CSD System Modules

5.3.1 Simulation Engine Module

The simulation engine is the primary component of a special purpose simulation tool. It determines the capabilities and constraints of the overall application. In a dewatering context, this module is responsible for the calculation of the water table level given the properties of the construction site and the pumping rates of the dewatering wells. A steady state model was chosen as the basis for the development of the simulation engine. This was deemed sufficient for demonstrating the capabilities of the application. Substituting more powerful engines such as MODFLOW is possible with minor changes to the existing structure and the development of further constructs to represent the additional supported capabilities. For the benefit of readers not familiar with hydrological terminology and steady state modeling, an overview of dewatering fundamentals is provided next.

5.3.1.1 Governing Hydrological Equations

An aquifer is a zone of soil or rock (geological unit) through which ground water readily moves. An aquiclude is an impermeable geological unit that is defined to transmit no water at all. An aquifer is normally classified as being unconfined or confined. The unconfined aquifer, shown in Figure 5-2(a), also known as a water table aquifer, is bounded beneath by an aquiclude, but is not restricted by any confining layer above. The confined aquifer, shown in Figure 5-2(b) is bounded above and below by an aquiclude.

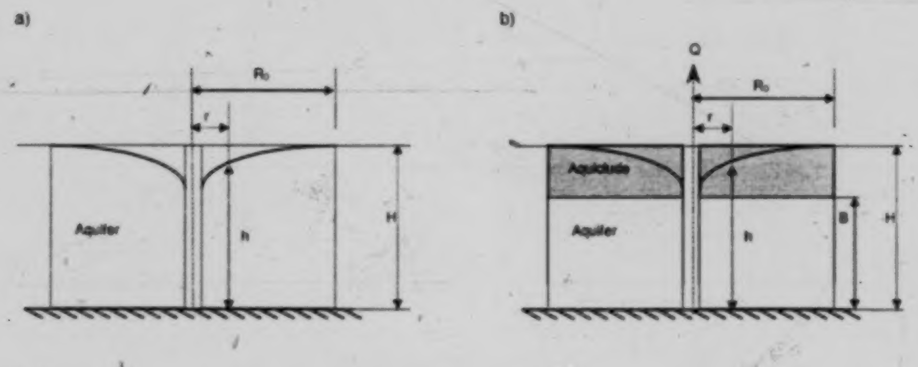


Figure 5-2 Typical Aquifer Types: a) confined; and b) unconfined

The basic equilibrium relationships that relate drawdown to pumping rates were presented by Muskat (1953) and Thiem (1906) as shown in equations [1] and [2]

[1] $$D = H - \sqrt{H^2 - \left(\frac{Q}{\pi \cdot K} \times \ln\left(\frac{R_0}{\cdot r}\right)\right)}$$ Unconfined aquifer.

[2] $$D = \frac{Q}{2\pi\, K\, B} \times \ln\left(\frac{R_0}{r}\right)$$ Confined aquifer.

Where

D: Water drawdown = H - h

H: Original water level (also know as phreatic surface)

h: Water level at radius r from the center of a pumping well

K: Effective permeability

Q: Pumping rate

R0: Radius of influence

B: Layer thickness

r: Distance between a point and the pumping well

The following assumptions apply: the well is pumped at a constant rate; the aquifer is homogeneous and isotropic with an infinite horizontal extent; the well fully penetrates the aquifer; and water is released from storage in the aquifer or other underground material in immediate response to a drop in water table level.

In the case of a multi-layered aquifer, the effective permeability can be calculated under the previous assumptions using equation [3].

[3] $$K = \sum \frac{K_i B_i}{B}$$

Where:

K: Effective permeability

Ki: Permeability of layer i

Bi: Thickness of layer i

B: Overall thickness of aquifer

5.3.1.2 Multi-Well Analysis

Another important concept is the cumulative drawdown principle. Essentially, if the drawdown in the aquifer is a small percentage (about 10-20%) of the aquifer thickness, the effect of each well can be superimposed on the other to determine the cumulative effect. This also applies to unconfined aquifers. Therefore, for multi-well systems, drawdown is given by equations [4] and [5].

[4] $$D = M \times H - \sum_{J} \sqrt{H^2 - \left(\frac{Q_j}{\pi \cdot K} \times \ln\left(\frac{R_j}{r_j} \right) \right)}$$ j=1 to M

Unconfined aquifer.

[5] $$D = H - \sum_{J} \left[H - \left(\frac{Q_j}{2\pi \cdot KB} \times \ln\left(\frac{R_j}{r_j} \right) \right) \right]$$ j=1 to M

Confined aquifer.

Where:

M: Number of pump wells

H: Original water table level

K: Effective permeability

B: Layer thickness

Qj: Pumping rate of well j

Rj: Influence radius of well j

rj: Distance between pumping well j and the observation point

5.3.1.3 Simulation Engine Implementation

The described mathematical formulation is implemented as a computer function which accepts as inputs all relevant data such as site characteristics (original water table level, aquifer type, depth, and effective permeability), pumping wells characteristics (influence radius, pumping rate and position within the site), and the position of the observation point. The function's output is the predicted water table level at the observation point.

5.3.2 Modeling Module

The modeling module encapsulates all data provided by the user and provides a graphical user interface for the definition, manipulation and viewing of this data in a variety of formats. This module was designed using an object-oriented approach under an event driven graphical user interface. Construction of a dewatering model is performed by incrementally adding objects and defining their properties. For example, pumping wells are added by selecting the appropriate icon from a toolbar and then selecting the desired position within the site. Any object on the screen can then be manipulated using standard "click-and-drag" techniques to modify its position or by "double-clicking" to access its graphical dialog box. A sample model definition is shown in Figure 5-3. The main toolbar at the top of the figure is used to access support functions such as saving, loading and printing. The secondary toolbar below is used to select the type of object to create.

The following subsections will provide a brief overview of the main modeling module classes and their functions.

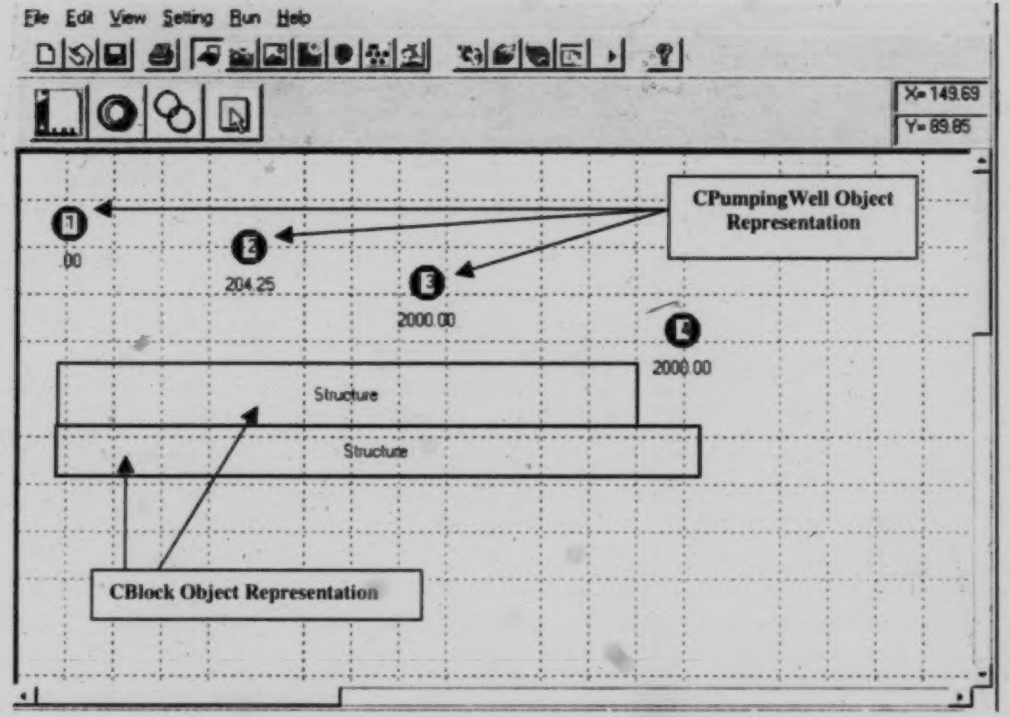
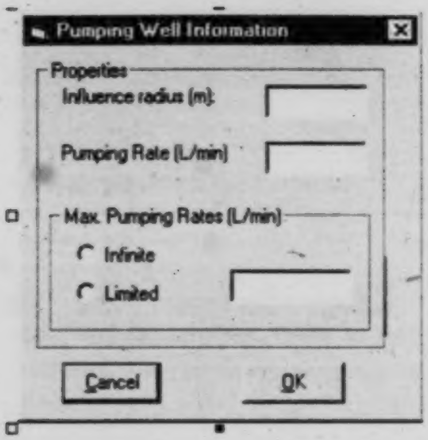

File Edit View Setting Run Help
X= 149.69
Y= 89.85
CPumpingWell Object Representation
1
.00
2
204.25
3
2000.00
4
2000.00
Structure
Structure
CBlock Object Representation

Figure 5-3 Main Object Definition and Manipulation View

5.3.2.1 CPumpingWell

This class encapsulates the properties of a pumping well and provides graphical manipulation capability through the dialog box shown in Figure 5-4. The pumping well properties are the influence radius (Rj), the well pumping rate (Qj), and the capacity (Cj). The capacity (or maximum pumping rate) of a well is used for the optimization module, which will be discussed later in this paper.



Pumping Well Information

Properties

Influence radius (m):

Pumping Rate (L/min)

Max. Pumping Rates (L/min)

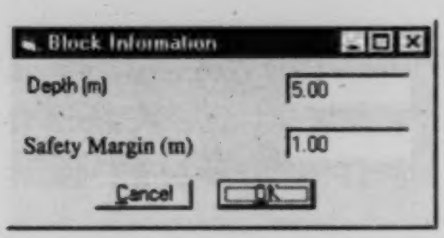
○ Infinite

○ Limited

Figure 5-4 CPumpingWell Class Dialog Box

5.3.2.2 CBlock

The "CBlock" object represents a three dimensional construction excavation area within the dewatering site. The length and width of the block object are defined by first selecting the "CBlock" icon from the secondary toolbar previously shown in Figure 5-3, then defining the affected area by clicking and dragging the mouse. The "CBlock" dialog box shown in Figure 5-5 can then be used to manipulate the other properties consisting of the depth of excavation and the safety margin. The safety margin is used for optimization and will be discussed later.



Block Information

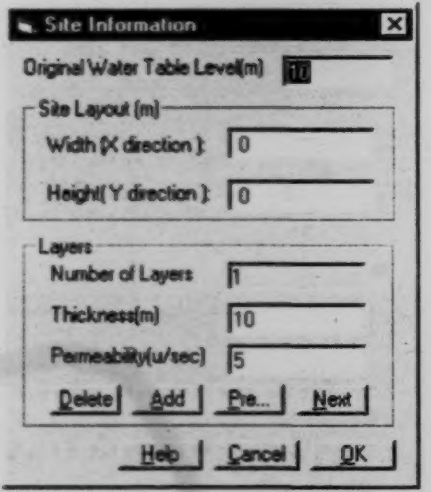
Depth (m) 5.00

Safety Margin (m) 1.00

Figure 5-5 CBlock Class Dialog Box

5.3.2.3 CSite

This class encapsulates site level information and acts as the coordinator between all other classes. Site level information, including layer data is defined through the graphical dialog box shown in Figure 5-6. General site data consists of the overall site dimensions as well the original water table height (H). Layers are added and deleted using the bottom portion of the dialog box. Layer data consists of the thickness (Bi) and the permeability (Ki). "CSite" acts as a "container" class for the "CPumpingWell" and the "CBlock" objects and provides functions for the creation, manipulation and deletion of these objects.



Original Water Table Level(m) 10

Width (X direction): 0

Height(Y direction): 0

Number of Layers 1

Thickness(m) 10

Permeability(u/sec) 5

Figure 5-6 CSite Class Dialog Box

"CSite" also provides graphical display functions to represent data in a variety of formats. The first is the standard editing view previously shown in Figure 5-3. The second view allows users to examine a horizontal or vertical cross-section of the site at any point. Figure 5-7 illustrates a vertical cross-section across pump 3 from the model shown in Figure 5-3. The third view is a contour representation of the water table level while the fourth view is a three dimensional representation which can be used for presentation purposes.

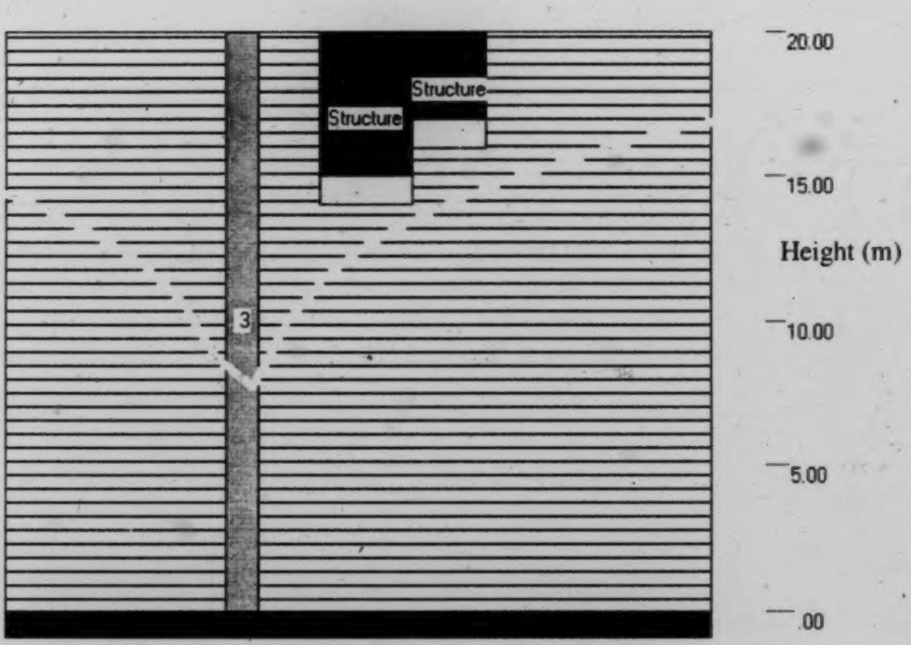


Figure 5-7 CSite Cross-Sectional View

5.3.3 Translation Module

The translation module handles the communication between the the modeling module and the simulation engine. It performs this function by first extracting the data from the object-oriented representation, invoking the appropriate simulation routines with parameters representative of the extracted data, then passing the results back to the modeling module. Although this module is shown as a separate logical entity in Figure 5-1, it is, in fact, implemented as another set of member functions of the classes presented in the previous section. Translation is performed when the user requests that the module be simulated by selecting "run" form the main menu. A pseudo code format of the translation algorithm is as follows:

1) Initialize a two-dimensional array to represent the water table level throughout the site at user-defined discrete intervals.
2) Calculate the effective permeability (K) using equation 3 for all layers contained in the "CSite" object.
3) Collect the pumping well information from all "CPumpingWell" objects.
4) For each array element initialized in step 1 representing a geometric position within the site, calculate the water table level by invoking the simulation engine function with the parametres obtained from steps 2 and 3.

At the end of the fourth step, the two dimensional array contains a representation of the water table level throughout the site. This information is passed back to the "CSite" object where it is used to initialize the cross-sectional, contour and three-dimensional views.

5.3.4 CSD's Integration Support

Integration support was identified as a key feature of the system in order to reduce data entry and redundancy and automate certain tedious processes. To illustrate the application of integration principles to the developed application, an optimization module was developed and linked to CSD. The purpose of this module was to provide the user with an alternative to iterative what-if analysis. Without automated optimization, the user is forced to manually set the desired pumping rates, perform the analysis, and then exam the results to ensure that the predicted water table level lies below the required excavation level.

The main objective of a dewatering analysis is to obtain the minimum pumping rates that satisfy given drawdown requirements throughout the excavation area. Aziz et al. (1989) presented a computerized method for determining the optimal pumping rates that meets user-defined drawdown requirements. In their model, these requirements are specified using the notion of a "watch well". The watch well represents a constraint on the water table level at a single position within the dewatering site. Equation [6] shows the system of equations used to predict the drawdown at the specified observation or "watch well" for an unconfined aquifer.

$$D_i = M \times H - \sum_j \sqrt{H^2 - \left(\frac{Q_j}{\pi \cdot K} \times \ln\left(\frac{R_j}{r_{ij}} \right) \right)} \qquad \text{i=1 to N and j=1 to M} \tag{6}$$

where

N: Number of observation points

Di: Drawdown at observation point i

rij: Distance between observation point i and well j

The system of equations is then solved and the resultant solution vector represents a set of pumping rates that minimizes the distance between the predicted and desired water table level. One of the drawbacks of this model is that it produces negative pumping rates for certain arrangements of wells and requires that the number of observation points be the same as the number of pump wells.

To overcome these limitations, an alternative mathematical optimization model was developed. The objective function minimizes the pumping rate of the entire system while satisfying the desired drawdown requirements. This method defines the model in such a manner that the predicted drawdown at the observation points can meet or exceed the desired level. Further, there are no limitations on the number of observation points. The mathematical model is non-linear in the unconfined case and linear in the confined. The complete mathematical formulation is as follows:

Minimize $\sum Q_j$

subject to

$$\left\{ M \times H - \sum_j \sqrt{H^2 - \left(\frac{Q_j}{\pi \cdot k} \times \ln\left(\frac{R_j}{r_{ij}}\right)\right)} \right\} \geq D_i$$

For i=1 to N and j=1 to M

If aquifer is unconfined

$$\left\{ H - \sum_j \left[H - \left(\frac{Q_j}{2\pi \cdot kB} \times \ln\left(\frac{R_j}{r_{ij}}\right)\right) \right] \right\} \geq D_i$$

For i=1 to N and j=1 to M

If aquifer is confined

$Q_j \geq 0$ For j=1 to M

$Q_j \leq C_j$ For j=1 to M

Where:

M: Number of pumping wells

N: Number of watch wells

H: Original water table level

K: Effective permeability

B: Aquifer Thickness

Cj: Capacity of well j (maximum pumping rate)

Qj: pumping rate of well j

Rj: Influence radius of well j

rij: Distance between "watch well" i and pumping well j

Di: Desired drawdown at "watch well" i

Implementation of the optimization module and its integration code required the definition of additional properties for the "CPumpingWell" and the "CBlock" classes. For the "CPumpingWell" class, the maximum pumping rate property was added to allow users to specify a maximum limit which acts as a constraint on the optimization model (Cj). The safety margin property was added to the "CBlock" class to allow users to extend the drawdown requirement beyond that implied by the excavation depth. The combination of excavation depth and safety margin are used to obtain the overall drawdown requirement (Di).

When optimization is requested by the user, a second translation module is initiated to prepare data to be passed to the optimization engine. The algorithm for this module is summarized as follows:

1) Create an array of watch wells based on the definition of the "CBlock" objects. Note that each "CBlock" object could result in the creation of numerous watch wells depending the area covered and a user-defined increment. For example, if the length of the "CBlock" object is 10 metres, the width 5 metres and the increment set at 1 m, the number of watch wells created is 50. The drawdown requirement of each generated watch well is generated from the sum of the depth of excavation and the safety margin.
2) Calculate the effective permeability (K) using equation 3 for all layers contained in the "CSite" object.

3) Collect the pumping well information from all "CPumpingWell" objects.

4) Pass the information obtained from steps 1, 2 and 3 to the optimization engine and initiate optimization.

5) Upon successful completion of step 4, obtain the optimal pumping rates and pass back to the "CpumpingClass" objects.

The optimization engine uses the Generalized Reduced Gradient Method (Lasdon et al. 1978) nonlinear optimization algorithm to solve the formulated mathematical model.

5.4 Case Study

CSD is used to complete a dewatering analysis for a project. The objective of the first experiment is to verify the accuracy of the steady-state model. The second analysis is performed to demonstrate the advantages of using a special purpose simulation environment for decision making.

5.4.1 Project Information

Hydrogeologic data obtained from the site indicated the presence of a water table 11 meters below the surface. Further, a silty-sand aquifer and fine to medium sand aquifer was identified from the soil borings. The sand aquifer was found at a depth of 13 meters below the surface. The top 13 meters consisted of a clay layer. Two pumping tests were conducted for the sand aquifer; they provided two important parameters: Transmissibility (T) = 6.8 X 10 -4 m2/s and Storativity (Cs)= 5 X 10-4. A summary of this information is shown in Figure 5-8.

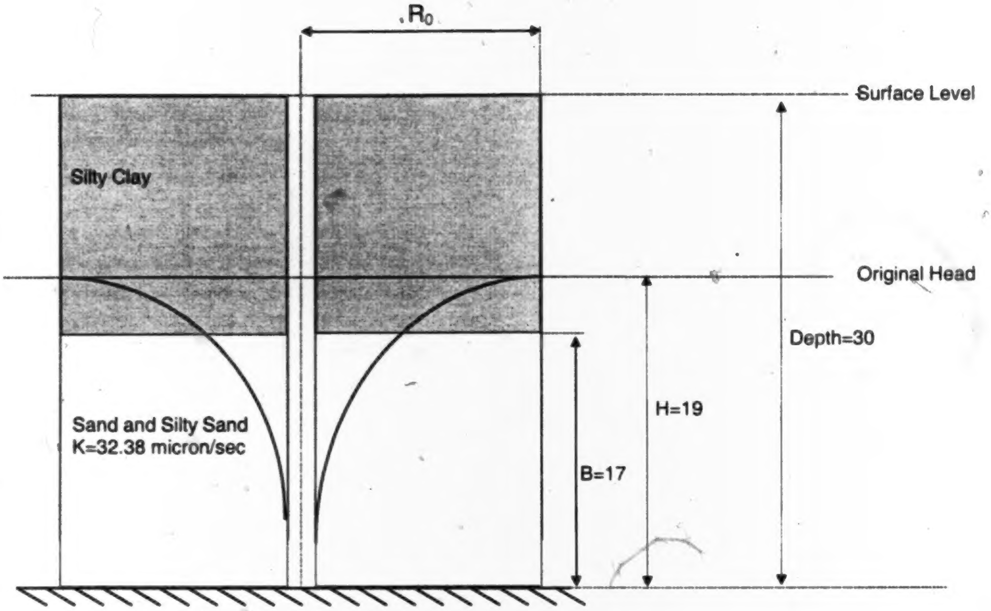

R0
Surface Level
Silty Clay
Original Head
Depth=30
Sand and Silty Sand
K=32.38 micron/sec
H=19
B=17

Figure 5-8 Soil profile in excavation area for case study

The construction project required that the water table be lowered in order not to interfere with excavation activities. A wellpoint system consisting of 23 wells penetrating to a depth of 30 metres was used as shown in the project layout in Figure 5-9.

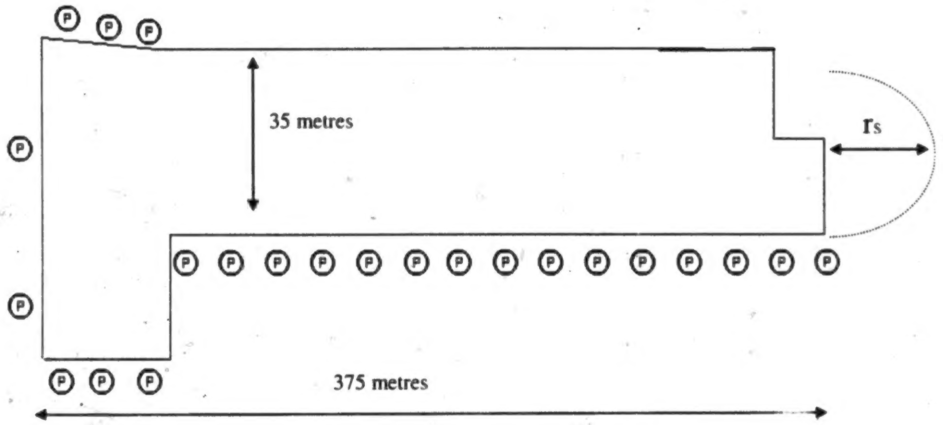


Figure 5-9 Pumping well layout for case study

Piezometer data taken before and during the pumping operation indicated that the water table on the northeast part of the project was lowered by an average of 3.7 meters within 24 hours. Total pumping rate of the system was approximately 1,400,000 litres per 24-hour day.

5.4.2 Simulation Engine Validation

General project information is entered as stated in the project specifications and as determined from the hydrogeologic data. The model uses the confined drawdown equations since the clay layer is relatively impermeable. Twenty three pumping wells are then added in the arrangement shown in Figure 5-10. The permeability of the sand aquifer is calculated as follows:

$$K = \frac{T}{B} = \frac{(1{,}000{,}000)(0.00068)}{17} = 40.0 \text{ micron per second}$$

where

T	Transmissibility
B	Sand Aquifer thickness

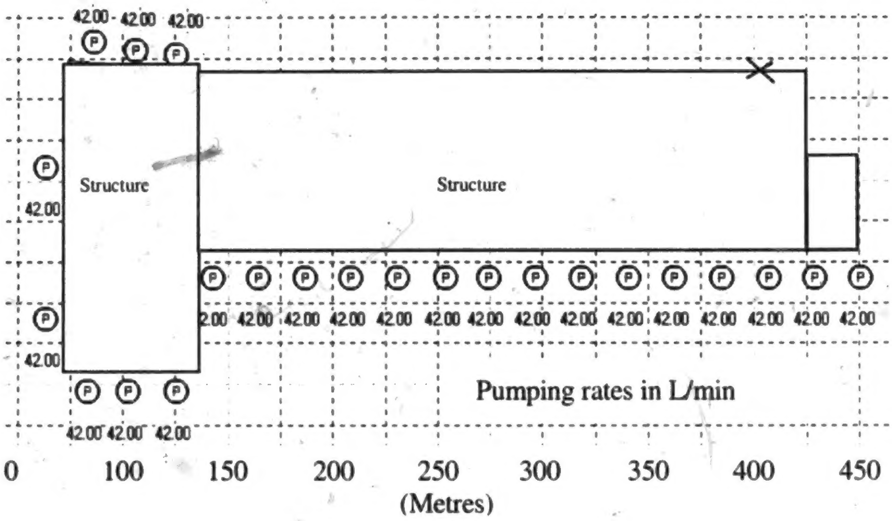

Structure
Structure
42.00
P
Pumping rates in L/min
0
100
150
200
250
300
350
400
450
(Metres)

Figure 5-10 Representation of Case Study Data in CSD

Project documentation did not record the water flow from each pump. This is understandable since all the pumps are usually linked into a single discharge line. To account for this, it is assumed that during the dewatering operation, all pumps were operating at the same maximum rate. Since the actual total pumping rate was 1,400,000 litres per day for 23 pumps, the pumping rates of each pump is assumed to be approximately 42 litres per minute.

As indicated by Equations 1 and 2, the influence radius is an important parameter of the equilibrium equations. Powers (1981) explains how an adapted form of Jacob's formula can be used to obtain an accurate estimate.

$$R_0 = r_s + \sqrt{\frac{Tt}{C_S}} + R_1$$

[7]

Where

R0	Radius of influence
rs	Equivalent radius obtained by assuming the entire system acts as a single large well. In this case, rs is estimated to be 20 metres (see Figure 5-10).
T	Transmissibility
t	Time available to achieve steady state.
Cs	Storativity constant
R1	Adjustment for possible recharge into aquifer.

By substituting the known values into the equation, the following is obtained:

$$R_0 - R_1 = 20 + \sqrt{\frac{(0.00068)(86400)}{(0.0005)}} = 363 \text{ metres}$$

The estimated recharge constant (R1) cannot be obtained from the information provided in the survey data. Further testing and surveying of the outlining areas is required in order to determine the possible effects of various recharge sources such as rainfall, snowmelt, nearby rivers, and sewers. For the purpose of this paper, R1 is assumed to have no effect (i.e. Radius of influence= 363 metres).

After the construction of the CSD model, the analysis was performed. Then, the water table level at the point of interest was compared to the actual water table level. By moving the computer mouse to the location marked by "X" (see Figure 5-10), the predicted drawdown can be obtained. CSD predicts a value of 3.95 of metres as compared to an actual value of 3.70 metres. The 6% error could be due to inaccuracies in the hydrogeologic data or lack of adjustment for recharge. In the absence of complete data, a 6% error is considered acceptable.

By using CSD's powerful graphing capabilities, the water table throughout the site can now be viewed. Figure 5-11 overlays the contour plot over the site layout. The values indicate the water table level as measured from the bottom of the aquifer.

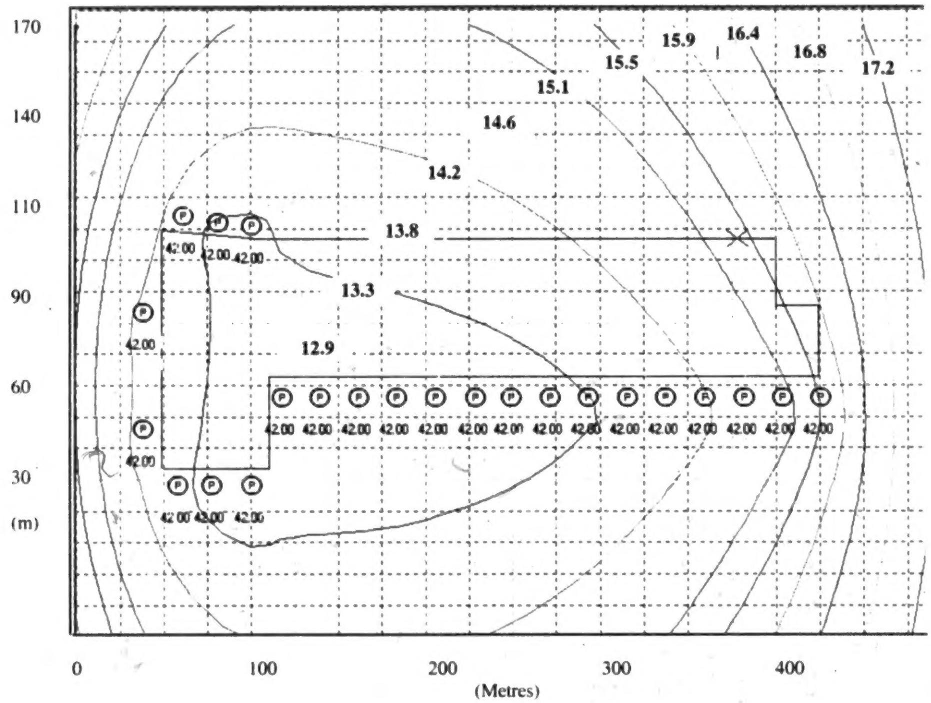


Figure 5-11 Contour-View of Water Table Level Throughout Site for Case Study

5.4.3 Demonstration of CSD's Advantages in Decision Making

The advantages of using an integrated optimization model are now demonstrated. The same information used in the previous section is used to examine whether all pumps should be operating at the given pumping rate. The objective is to minimize the total pumping rate of the system to order to minimize discharge costs. The other objective is to minimize the number of pumps in order to reduce drilling and operating costs.

The optimization model requires the capacities of the pumps (i.e. maximum pumping rate). All these values are set to the observed operating level of 42 litres per minute. After performing a mathematical optimization, CSD predicts that two wells in the west side of the area should have a pumping rate of zero indicating that they are not required. Furthermore, another set of wells is operating below the maximum capacity. According to CSD, the total pumping rate of the system would have been reduced to 847 litres per minute or 1,220,000 per day if the suggested optimized setup was used.

5.5 Conclusions

A computer application has been presented for the analysis of construction site dewatering projects. The tool was validated as part of a detailed case study. Several features allowed CSD to be successfully implemented. In particular, it was observed that any simulation tool for dewatering type applications should adhere to the following guidelines:

1. Users should not be required to posses strong expert knowledge in the field of hydrology to use the system. This means that the modeling environment should allow for the creation of models using simple and basic constructs such as wells, layers, and excavation areas.
2. Users should be able to build models based on site data obtained from standard hydrological surveys.
3. The modeling environment should be user-friendly, intuitive and highly graphical, allowing for the construction of dewatering models in a relatively short period of time.

4. Analysis results should be of relevance to a construction practitioner. This is especially crucial if the results are to be used for cost-estimating purposes in a tendering environment.

Chapter 6 – Phase One Research Findings[1]

6.1 Introduction

The previous three chapters presented the successful application of computer simulation technology to three distinct construction applications. This research demonstrated the applicability, effectiveness and practicality of using simulation theory for planning construction related projects. These case studies lead to the formalization of the basic requirements that such tools must posses in order to be successfully implemented in an industry setting. Their limitations also helped in identifying further issues, which would have to be addressed to order to increase their effectiveness. This chapter will discuss both these requirements and limitations in general.

The identified basic requirements can be grouped into five categories and will be presented in separate sections (section 6.2 through 6.6). An analysis of the observed limitations is provided in section 6.7 .

6.2 User Interface

The user interface plays a large role in determining the usability of the tool. A graphical user interface helps in explaining the structure of the developed models and makes it possible for people with minimal computer skills to use the tools. Specifically, it was observed that the user interface should have the following properties:

---

[1] A version of this chapter was published as "A Framework for Applying Simulation in the Construction Industry" in the Canadian Journal of Civil Engineering, Volume 25, Number 3, June 1998.

1. Visual iconic elements for describing the model. The modeling elements should use graphical representations of objects within the domain (e.g. Road in AP2-Earth, crusher in CRUISER and a pumping well in CSD). A close mapping between the actual system and these icons is essential.
2. A free format for modeling which resembles a "sketching board" approach on a computer screen. In AP2-Earth the user is free to specify roads, source locations, dump locations, equipment etc. in a flexible manner which does not force her to follow a rigid framework or series of structured steps. Following these steps results in simple model building process that is non-linear and extremely flexible in terms of object manipulation. Users are able to freely navigate between the various stages of model construction without any limitations.
3. A user-friendly and robust interface providing error detection and facilitating tracking of logical problems within the model.

6.3 Modeling Philosophy

Modeling is the process of defining the various simulation components, their properties and their interactions. The structure of the model determines the flow of the underlying simulation code. It was observed that users must not be exposed to the underlying simulation model constructs. The tools should abstract these constructs and provide a high level environment where modeling can be done in a natural manner. For example, a traffic interaction situation in AP2-Earth is supported through the “Intersection” modeling element. This element encapsulates the low-level simulation resources, entities and events, which are required to model an intersection. Validation of model properties and structure should also be performed in the user-friendliest fashion. This means that errors should be reported as soon as they are detected. For example, providing an invalid crusher setting in CRUISER should be trapped and reported immediately through a dialog box.

6.4 Simulation Execution Approach

The simulation engine is responsible for processing the simulation code generated by the modeling environment. The three developed tools employ different type of engines.

AP2-Earth uses a discrete-event simulation algorithm where events are processed based on the defined activities and the logical relationships between process components and availability of resources. Systems modeled using this approach are dynamic in nature. The translation of the visual earth-moving model into data that is usable by the engine is the responsibility of the modeler. An example of how this is done for the hauling process is demonstrated in Figure 6-1.

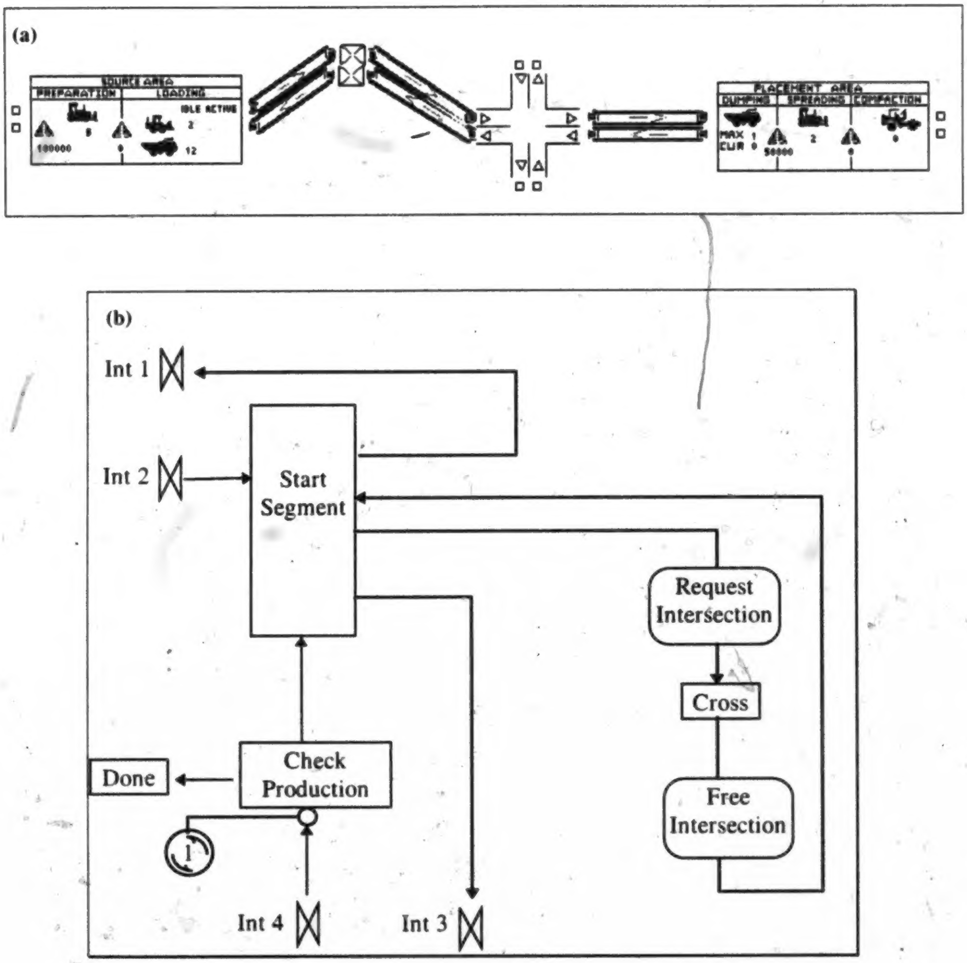

(a)
SOURCE AREA
PREPARATION
LOADING
IDLE ACTIVE
100000
2
12
PLACEMENT AREA
DUMPING
SPREADING
COMPACTION
MAX 1
CUR 0
50000
(b)
Int 1
Int 2
Start Segment
Request Intersection
Cross
Free Intersection
Check Production
Done
1
Int 4
Int 3

Figure 6-1 Translation of High Level Model to Discrete Event Representation

CRUISER uses a static simulation algorithm, which is driven by prescribed processing flow that is not dependent on time or interaction of resources. The objective of the simulation engine is to sequence a variable number of analysis operations with dynamic relationships. The approach is similar to AP2-Earth except that the algorithm is different. The visual model is first translated into useful data for the engine to process. The data primarily represents what equipment and settings are used and how the material flows from the source to the end product pile. The processing is performed using an event queue. The algorithm simply determines the next event or tasks, which must be performed in accordance with the user defined plant layout. Each scheduled event holds information pertaining to the type and identity of the equipment being modeled. Detailed implementations were discussed in section 4.3.9.

In CSD, the user-defined model is first translated to the equivalent mathematical representation by mapping the visual components into the equivalent mathematical model. Next, the resultant system of equations is evaluated. Finally, the results are provided using various reports that can be accessed from the modeler. A sample report was previously illustrated in Figure 5-11.

6.5 Simulation Result Representation

Utilizing simulation tools is an iterative process where models are built, simulated and continually refined to achieve the desired results. Simulation results provide the information needed to evaluate a given simulation experiment. It has been observed that the content, format and presentation of results must be tailored to the specific domain of the tool. For example, AP2-Earth employs a list of standard and detailed reports that can be used to quickly analyze the overall performance of the model in terms of activity productions or detailed results such as individual queuing statistics at intersections. Reports in CRUISER and CSD use formats that appeal to their respective industries. The CRUISER reports provides information regarding the gradation of the "product" piles, which is significant for operations of general pit mines. The CSD reports include chart representation of the final water table level resulting from use of the pumps determined by the simulation engine.

6.6 Integration Requirements

A great factor in the successful implementation of all tools at construction companies was their capability to integrate with existing systems. This reduced the overall data entry requirements and allowed the simulation results to be viewed directly as part of systems that users were already accustomed to. For example, in order to make AP2-Earth appealing to an earth-moving contractor, it was given the capability to generate information for use by a cost estimating module.

6.7 Conclusions

This chapter outlined the key features of three specialty simulation tools that led to their successful implementation with collaborating construction companies. The next phase of the research involved an initial assessment of further needs in construction simulation. These needs, combined with the identified features, eventually lead to the development of the unified modeling methodology, which will be presented in the next chapter. The results of the analysis of the three developed systems revealed the following issues:

- Modeling of large construction projects, which include a multitude of fundamental construction operations, cannot be easily performed solely with standalone specialty tools. There is a need for a concept that can combine models based on several tools.
- Simulation model development is a time-consuming activity, even when specialized tools are used. There is a need for an approach that heavily supports and even encourages the reusability of constructed simulation models.

- Simulation tools require a large amount of input data. Some of the input data can only come from the user's knowledge; the remainder can typically be retrieved from other company systems including historical databases, accounting systems, equipment databases and CAD systems. An approach is required that can provide seamless support for integration with other company systems.
- Simulation tools also generate a large amount of useful results. There is a need to standardize the content and the format of the generated output in order to maximize its effectiveness and utilization in planning construction projects.
- Large amounts of expertise, experience, and time are required for development of new specialty tools. An approach is needed whereby the development of these tools is greatly simplified. This will greatly minimize the amount of the initial investment and increase the use of simulation in general.

Chapter 7 – Unified Modeling Methodology

7.1 Introduction

A unified modeling methodology has been developed to address both the requirements identified in the first phase of the research and the further identified issues. This methodology, illustrated in Figure 7-1, unifies several state-of-the-art concepts with other concepts developed as part of this thesis.

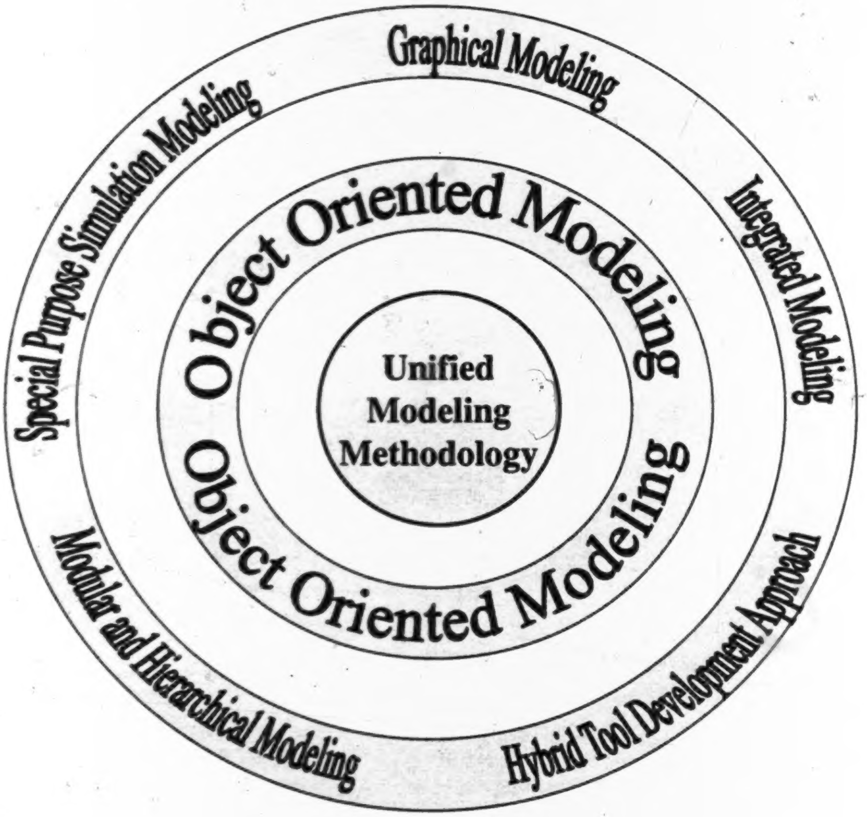

Graphical Modeling
Integrated Modeling
Hybrid Tool Development Approach
Modular and Hierarchical Modeling
Special Purpose Simulation Modeling
Object Oriented Modeling
Object Oriented Modeling
Unified
Modeling
Methodology

Figure 7-1 Unified Modeling Methodology and Contributing Concepts

The concepts and methods followed in making the unified modeling methodology work, will be discussed in the rest of this chapter. Section 7.2 through 7.7 will discuss each contributing concept individually. Section 7.8 will then discuss the approach followed for their unification.

7.2 Special Purpose Simulation Modeling

7.2.1 Introduction

With general-purpose simulation (GPS), simulation models are constructed using a set of abstract building blocks such as queues, activities, branches, and statistical collection nodes. An example model of a simple construction operation built with CYCLONE elements is shown in Figure 7-2. Simulation results are limited to queuing statistics, average waiting times and resource utilization. Any desired custom statistics must be explicitly declared using the existing statistical collection nodes. Also, in numerous cases, further modeling is required in the form of programming code inserts. This is due to the fact that even with abstract building blocks, there is still an inherent level of expressive limitation.

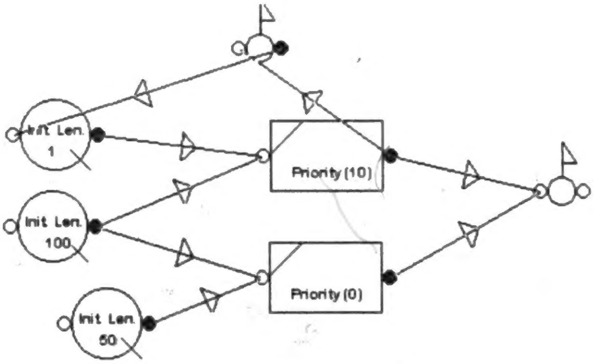


Figure 7-2 Sample CYCLONE Simulation Model

GPS provides a highly expressive modeling environment but forces the modeler to think in abstract terms. Developed models consist of a logical description of the process without explicit mapping to real life components. This feature of GPS coupled with the frequent requirement to write programming code means that simulation is mainly limited to the academic and expert arena.

It has been discovered through industry collaboration and research that the modeling environment must be tailored to the specific requirements of the intended construction domain. This led to the development of the alternative concept of special purpose simulation modeling (SPS). Through specialization, a simulation tool's usability outside its intended scope becomes limited. However, the benefits far outweigh any loss in flexibility. Computer tools conforming to the ideas of SPS enable a practitioner who is knowledgeable in a specific construction domain, but not necessarily in simulation, to model a project within that domain in a manner where graphical representations, navigation schemes, specification of model parameters, and representation of simulation results are completed in a format native to the domain itself.

The underlying assumption with SPS is that a knowledgeable developer can develop a set of specialized modeling elements or building blocks for a given domain which are easy to understand and flexible enough to allow someone else to construct simulation models in that domain. The ability to model different scenarios is supported by the fact that the developed modeling elements support the concept of parameterized modeling and flexible layouts. This means that the generated simulation code is based on the defined layout as well as the supplied parameter values. This assumption is easily satisfied in construction simulation because a great deal of construction processes can be well scoped and specified for the purpose of a SPS tool. Examples of processes already investigated in the first phase of this research are earth-moving, aggregate production, and dewatering operations. Other processes that have been identified as potential candidates for a SPS application are road works, paving and tunneling operations.

7.2.2 SPS principles

Research from the first phase identified three fundamental requirements that a given simulation tool must satisfy to adhere to the principles of special purpose simulation.

First, the set of available modeling elements must map to real physical or logical components of the target process being analyzed. For example, an aggregate production plant model is built using a set of modeling elements that include crushers, screen decks, and sieve analysis nodes. With this approach, the modeler is no longer limited to the abstract set of modeling elements. Instead, a relevant and specialized set of modeling elements is available for building models in an intuitive and direct manner.

Second, the model parameters should be specialized and associated with the modeling element they correspond to. Parameters are properties of the model that the user can change to customize the behavior of the simulation. By associating parameters with the modeling elements, users can directly and intuitively navigate to the desired parameters in order to change their values. For example, in an aggregate production SPS tool, the user should be able to change the setting of a given crusher by accessing the parameters of the crusher modeling element rather than through some common global parameter listing. The data entry process associated with setting the parameter values should be as user-friendly and intuitive as possible. Different types of parameters such as numeric, text, Boolean and array should also be allowed. Validation of parameter values should also be performed as soon as possible to reflect model constraints.

The third requirement stipulates that all the potentially useful statistics and results must be generated and made available to the user through the modeling elements. With GPS, statistics and other results are presented together on one or more large reports. The drawback is that the user must navigate a long list of generated outputs in order to find the desired results. Further, some of the generated data will typically require further analysis. This lengthens the "What-If" analysis cycle and reduces the effectiveness of the tool. SPS stipulates that generated results must be in a format that is easy to navigate and is of immediate use for evaluating model performance. This means that no separate analysis should be required and the results themselves should be accessed through the associated modeling elements, which had a primary role in their production. Further, results must consist of the traditional GPS types of information such as averages, standard deviations, histograms and confidence intervals, as well as specialized information related to the domain. For example, a sieve analysis element in the aggregate production SPS tool should generate the results of the analysis as a gradation graph. An example of this graph is shown in Figure 7-3.

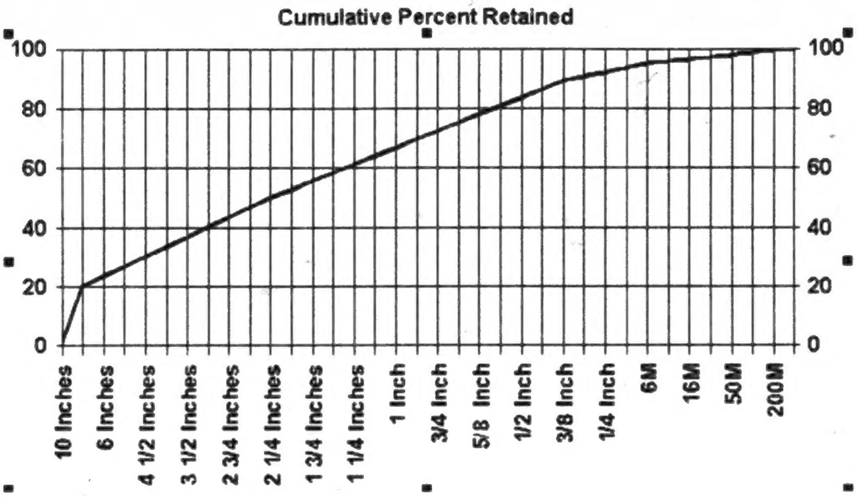


Figure 7-3 An Example of Generated Simulation Output in a Graphical Format

7.2.3 Model-Reusability through SPS

One problem that has been identified with traditional simulation systems is a lack of support for simulation model-reusability. The SPS approach directly supports the concept of model-reusability through the specialized modeling elements themselves. A SPS tool is merely a template of modeling elements, each of which encapsulates parameterized simulation code. The generated simulation code for the entire model, which is used to run the simulation, is a function of the model layout as defined by the user and the supplied parameter values. Each time the user builds a SPS model and runs it, the simulation model of the base modeling elements is, in fact, being reused.

7.2.4 SPS Limitations

Special purpose modeling does have limitations. First, the user is limited to the available set of modeling elements. This initial set of modeling elements is designed to be flexible and reusable within a target domain and subject to a defined scope. If a situation arises within that domain that falls outside the defined scope, the user must resort to complex workarounds or she may be unable to use the tool at all. This places a heavy burden on the SPS tool developer in foreseeing all potential scenarios, which is a very daunting if not impossible task. This is why SPS modeling is best used in combination with other concepts such as modular and hierarchical modeling, as will be explained later in this chapter.

7.3 Graphical Modeling

7.3.1 Introduction

A major component of simulation systems is their user interface. Text-based interfaces require the user to create and manipulate textual representations of the model. This requires knowledge of the embedded syntax and, while perfectly adequate for the experienced modelers, is generally difficult for novice users. Graphical user interfaces (GUI) simplify this process by allowing the user to view and manipulate a given model without having to resort to editing textual representations. With GUIs, the basic tasks are centered around both mouse and keyboard operations.

Graphical modeling combined with special purpose modeling results in highly intuitive user interfaces. Constructed models resemble a flowchart of the modeled process. This has the added benefit of allowing the graphical representation to be used as a means of communicating the construction method specification.

Graphical modeling includes both graphical manipulation and graphical representation. The next sections will provide guidelines on the use of a GUI for simulation modeling systems.

7.3.2 Graphical Model Manipulation

One aspect of graphical modeling is graphical model manipulation. This is done using an interface that resembles a drawing board where modeling elements are added, manipulated and related on a modeling layout window. The following is a list of basic manipulation activities:

- Addition and deletion of modeling elements through a toolbar or a menu.
- Manipulation of element positions by clicking and dragging the iconic representation of elements on the layout window.
- Manipulation of parameter values through an attribute dialog box, which is activated through a menu item or a mouse double-click.
- Manipulation of relationships graphically through mouse operations.

7.3.3 Graphical Representation

7.3.3.1 Modeling Element Representation

Modeling element are made to resemble the corresponding physical or logical real world element. For example, a jaw crusher in an aggregate production simulation tool will be graphically represented as a bitmap resembling a real jaw crusher. Similarly, a task in CYCLONE is made to resemble a square.

Representation of the relationships between modeling elements is also done graphically. Different SPS domains may require different types of relationship representation. For example, in an aggregate production type of simulation, relationships are represented graphically to appear as conveyors (Figure 7-4A), whereas in an earth-moving simulation, relationships are represented graphically as either logical arrows or as simple highlighted circles (Figure 7-4B).

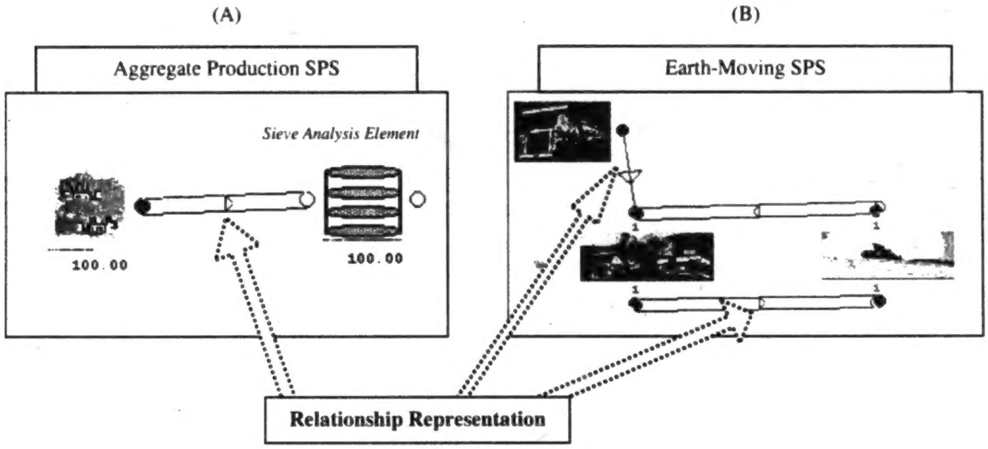

(A)
(B)
Aggregate Production SPS
Sieve Analysis Element
100.00
100.00
Earth-Moving SPS
1
1
1
1
Relationship Representation

Figure 7-4 Examples of Graphical Representation of Relationships in Two SPS Tools

7.3.3.2 Output representation

As part of the modeling element representation, certain state information can also be displayed. State information is based on the values of the supplied parameters, outputs or statistics. This helps the modeler in maintaining a good abstract view of the model during and after its development. For example, notice how in Figure 7-4A, The "Sieve Analysis" modeling element displays the feed rate underneath the bitmap representation to provide quick visual feedback to the user.

The complete list of available output and statistical analysis data is accessed through the element's attribute dialog box. Modeling elements generally generate numeric or textual data as well as tabular and graph-based outputs. For example, the "Sieve Analysis" element can generate a graph representing the product gradation. Statistical analysis results can also be displayed in a similar manner. With graphical modeling, collected observation can be displayed graphically as a histogram.

7.3.4 Issues with Graphical Modeling

Graphical modeling is an extremely beneficial concept for streamlining the modeling process. It is especially beneficial for novice users. It is mostly suited for relatively small models with a reasonable number of elements. As models become large their maintenance, through the GUI, becomes highly cumbersome and difficult to manage. This issue can be partly solved with the modular and hierarchical modeling concepts. A more powerful and flexible solution is to provide the model builder a facility to override the graphical user interface and resort to a supporting script-based interface if desired. To do this, a scripting environment must be supported wherein the user can both perform all the functions that the GUI performs using textual commands and have access to traditional programming structures, including conditional and looping statements. This concept is discussed in more detail in Section 8.6.

7.4 Integrated Modeling

Integrated modeling is a concept that defines how a simulation tool can access information in other systems to automate or support the model building process and generate information for use by other systems. The two main issues involved are the actual mechanics of the transfer mechanism, which can be handled through a database approach, and the nature and format of the exchanged data, which can be handled through data exchange standards.

7.4.1 Data Storage Format

Database management systems, as a alternative to flat files, have many advantages including support for multi-users, advanced data-based security, integrity checking, and centralized management of large amounts of information. Relational database management systems (RDBMS) are based on relational theory and are widely used in many areas of construction information systems, including cost estimating, scheduling and project control. Tapping into the wealth of information stored in these systems can greatly reduce the effort involved in building simulation models. This is can be accomplished by providing relational database access capability to the modeling elements themselves. Consider, for example, a truck element that requires the user to supply eight parameters that determine its travel speed given the total resistance of the haul road. Instead of requesting this information from the user each time, a standard table stored in a RDBMS can be automatically accessed. The user only has to select the truck type and the simulation system takes care of extracting the relevant information.

In addition, other systems should ideally be able to access the simulation results for automating their respective tasks. For example, a scheduling system should be able to extract activity duration from a simulation model. RDBMS can play a large role in making this possible. By storing the simulation models and results in the shared database itself, as opposed to a proprietary format, other applications can access this data directly. The RDBMS takes care of all the concurrency and access restriction issues.

An alternative method for sharing simulation data is through the concept of a documented object library. Object libraries expose a public interface to all data in a simulation model.

Other systems can access the methods and properties in this library to extract and even manipulate a given model. This method is discussed in further detail later in this chapter.

7.4.2 Standard for the Exchange of Simulation Data

7.4.2.1 Introduction

Traditional means of extracting data from simulation results for the purpose of creating project estimates or schedules involved hard-coded knowledge about the structure of this data. This meant that a separate interface had to be developed between each class of SPS tools and each external system. Further, typical simulation results often lacked potentially valuable information that could be generated for use by other support systems with minimal extra effort.

Efforts to integrate earth-moving and aggregate production models with estimating and scheduling systems have shown that a standard for data generation and exchange is required. This standard should specify not only the format of the transferred data but also the classes of data that should be generated from every SPS tool. This change in viewpoints shifts the emphasis from the external systems to the simulation tools themselves. The onus is then on the tool itself to deliver the appropriate information in the appropriate format. By following this strategy, a single extraction utility can be developed for each type of external system.

The developed standard defines how information should be generated in order to be beneficial for external project planning systems. The actual analysis of the raw data constitutes a body of research on its own and therefore, no attempt is made to define that part in this standard. The generated information consists of basic facts about the occurrence of planning-related events during the simulation. The interpretation and analysis of the data is the responsibility of external integrated systems.

7.4.2.2 Information Categories

Information generated by each SPS tool during simulation initialization or processing falls under the following categories:

- Activity Listing. This is a list of basic activities that will take place during simulation. This includes the activity name, unit of measure and any hierarchical relationships to other activities. This is the minimal amount of information required to construct an initial project breakdown for use in an estimating or scheduling system.
- Activity Resources. This is a list of resources for each activity that will be to be utilized. This information can be used to identify the type of resources that should be allocated for a given activity.
- Activity Cost Centers. This is a list of cost categories or centers for which cost will be incurred.
- Activity Schedule. This is a list of records that indicate the start and stop times of each activity. An activity can be stopped and started a number of times to signify fundamental interruptions. This information can be directly used by a scheduling system to obtain the total duration. It can also be used by an estimating system, which requires the duration for calculating indirect costs.
- Activity Production. This is a list of records indicating the quantity of work recognized for a given activity at a given simulation time. This quantity is provided in the unit of measure of the activity. The production information along with the

duration of the activity is used to obtain the productivity values, which drives the entire estimating process. The detailed production information available continuously over a period of time can also be used for material management systems. For example, a tunneling simulation tool might track its production based on the total quantity of earth removed over time. A material management system could access this information to provide a disposal subcontractor with a detailed spoil removal schedule.

- Activity Revenue. This is a list of records indicating the dollar value collected for a revenue generating activity at a given simulation time. This information basically provides a schedule of expected payments for the activity. This information, along with the costing information, aids cash-flow analysis systems used for planning the project and company finances.
- Activity Resource Utilization. This is a list of records indicating the change in utilization of a particular resource by a given activity at a given simulation time. Resource level information provides a report of the expected resource requirements of a particular activity as a function of time. This information can be used to optimize equipment and labor allocations and minimize their idle time.
- Activity Cost. This is a list of records indicating the cost incurred by a particular cost center on a given activity at a given simulation time. This information can be used directly by estimating systems or in association with revenue information for cash flow analysis.

7.4.2.3 Information Format

As mentioned earlier, the ideal format for information exchange is in the form of a relational database. This is mainly due to its features and wide use. As a result, the generated simulation planning information is represented relationally as shown in Figure 7-5.

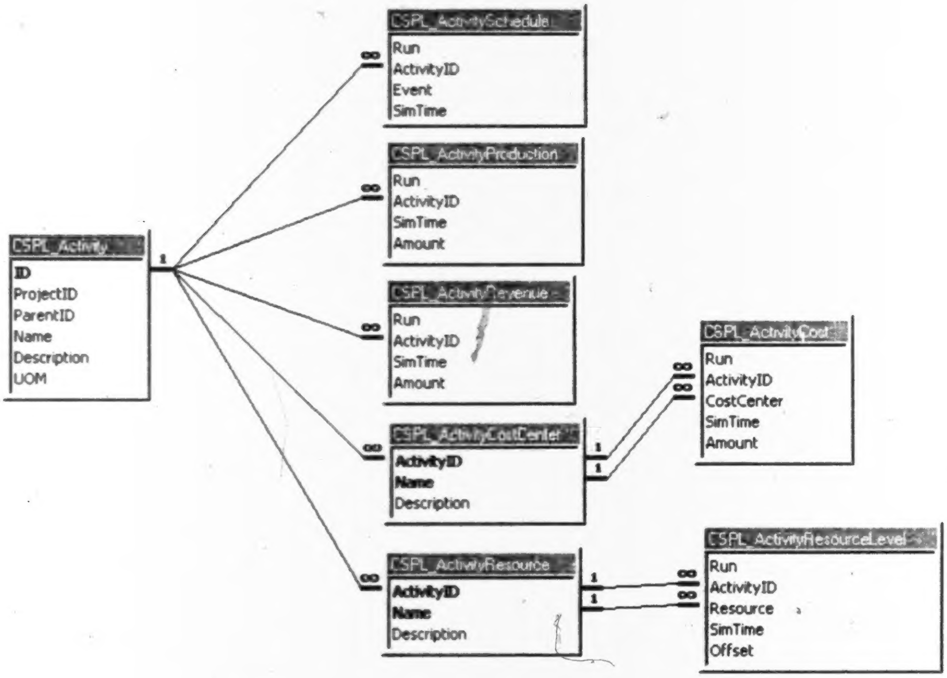

CSPL_ActivitySchedule
Run
ActivityID
Event
SimTime
CSPL_ActivityProduction
Run
ActivityID
SimTime
Amount
CSPL_Activity
ID
ProjectID
ParentID
Name
Description
UOM
CSPL_ActivityRevenue
Run
ActivityID
SimTime
Amount
CSPL_ActivityCost
Run
ActivityID
CostCenter
SimTime
Amount
CSPL_ActivityCostCenter
ActivityID
Name
Description
CSPL_ActivityResource
ActivityID
Name
Description
CSPL_ActivityResourceLevel
Run
ActivityID
Resource
SimTime
Offset
1
∞

Figure 7-5 Relational Representation of Generated Simulation Planning Data

7.5 Modular and Hierarchical Modeling

Modularity and hierarchy concepts allow for the modeling of large and complex projects. If used in combination with special purpose modeling, they allow for the linking of models based on modeling elements of different domains.

The concept of modularity defines a framework where simulation code can be packaged as a single unit and accessed through well defined input and output connection points. Input connection points receive entities from other packaged units, process a number of simulation events, then transfer the entity out through one or more output connection points. Multiple input connection points provide different entry points into the simulation code. They are used to generate multiple output entities or to support conditional and probabilistic branching. An example scenario with two input connection points and three output connection points is provided in Figure 7-6.

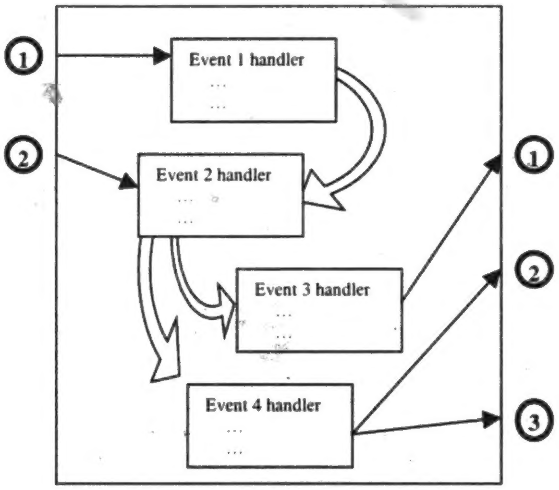


Figure 7-6 Use of Connection Points for Exposing Encapsulated Simulation Code

Modularity provides a benefit that is similar to the benefit of object-oriented languages: one modeling element can be built without explicit knowledge of the inner workings of the other elements. Entities traversing a given model through the connection points of the elements are analogous to the messages passed between objects in an object-oriented environment. Some elements will not have any input connection points. This usually indicates that they will generate entities themselves. Generated entities are routed using user-defined relationships between the connection points of the modeling elements.

With hierarchy, modeling elements can be created as children of other modeling elements. This allows large, complex models to be built using the divide and conquer approach. At the highest level, elements are created to represent high level operations such as substructure, superstructure and finishing. Inside each of these high level elements, further child elements are added to define the respective sub-operations taking place. At the lowest levels, detailed simulation models are defined to model the various fundamental construction processes. Elements that support the creation of child elements act in a manner similar to regular elements; they have ~~their~~ own input and output connection points. One important issue is how incoming entities arriving at the parent element should be routed to the appropriate child element. This is done by creating relationships between the input connection points of the parent and a given input connection point of a child element as well as relations between the output connection points of a given element child and the output connection point of the parent. An example of how this is done is shown in Figure 7-7. The parent element behaves just like any other element and can participate in relations with other elements, which may or may not contain child elements of their own.

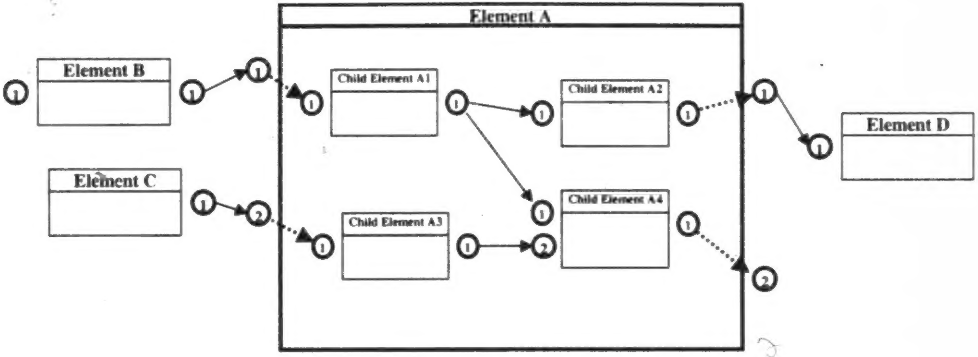


Figure 7-7 Utilization of Invisible Relationships for Routing Entities

The concept of hierarchy in simulation modeling can also greatly enhance the model definition process. Child elements can be used in an unconventional manner as a means of obtaining complex parameter data from the user. This supports the graphical modeling concept, which defines how simple textual, numeric and array type data are obtained. The mere existence of child elements with their own parameter values can be used as information that guides the simulation processing of the parent element.

Consider, for example, a crusher element that receives through its input connection point an entity representing the input stream and then generates an entity through its single output connection point a new entity representing the predicted stream after the crushing process. Crusher elements will contain a parameter indicating the desired setting of the crusher. For each possible setting, the crushing analysis algorithm requires a table that indicates the expected average performance of the crusher. This information must be supplied by the user. In computer terms, an array of numbers is supplied by the user for each possible setting of the crusher. A convenient way of having the user provide this information is to add a child element inside the crusher element called "setting". This setting element encapsulates the expected gradation table for a single setting. During simulation, the "crush" element will access the information from the appropriate child element depending on the user's desired crusher setting.

As with special purpose modeling, modular and hierarchical modeling leads to increased simulation model-reuse. While SPS encouraged model-reuse through flexible modeling elements that can be used in a variety of scenarios, modular and hierarchical modeling contribute to the regular model-reuse of existing elements or groups of elements. To illustrate this, consider an earth-fill dam construction project. It is evident that an earth-moving construction process will take place at several stages of the project. As a result, a single simulation model of an earth-moving process can be constructed and reused every time it is needed within the simulation model of the entire project. It becames quite simple to store these commonly used models in a user model library. which can be accessed for use in a completely different project.

Another added advantage of these concepts is that they allow for the linking of models developed with different SPS tools. Developers of SPS tools need only worry about the high level general purpose connection point, which will synchronize the two high level modeling elements representing the two processes. An example scenario of an earth-moving process feeding an aggregate production process is illustrated in Figure 7-8. At the highest level, a relation is established between two high level elements. The purpose of this relation is to inform the aggregate production model of the arrival of a new batch of earth following a truck cycle. Drilling down inside each high level element displays the detailed specifications of each respective model. Dotted block arrows have been added to illustrate the invisible relationships between child elements and the parent's connection points. These invisible relations are used to route entities as previously explained.

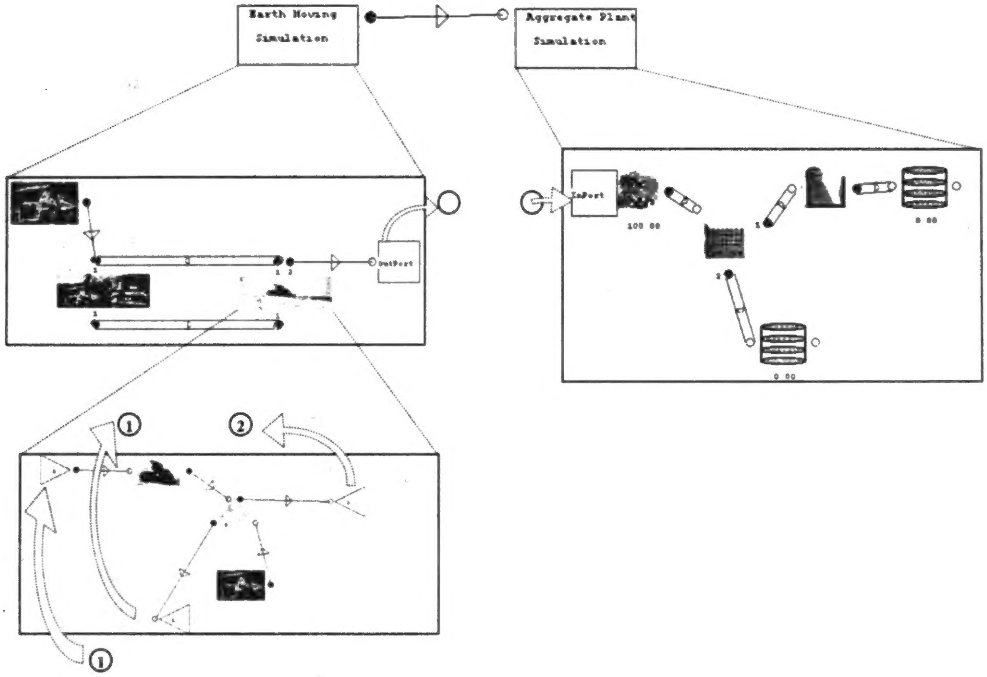


Figure 7-8 Use of Modularity and Hierarchy Concepts for Linking Multiple SPS Tools

When the concept of SPS modeling was discussed, it was pointed out that SPS templates can produce inflexible environments where the user is limited to modeling scenarios that fit the initial scope defined by the template developer. This limitation is easily alleviated by first developing an abstract template based on general purpose modeling paradigms such as CYCLONE, then using the modularity and hierarchy concepts described to allow users to supplement their specialized models with elements from the abstract template. An example of a model that utilizes abstract elements to model a regular maintenance operation within an earth-moving model is shown Figure 7-9. The added sub-model utilizes a branch type element to decide stochastically whether or not a truck should be maintained. If the truck does not need maintenance, it is sent directly to the output port element, which transfers the entity back to the earth-moving model. Otherwise, a maintenance crew resource is requested and the maintenance activity is executed with a stochastic duration. After the completion of the activity, the crew is released and the truck is transferred back out.

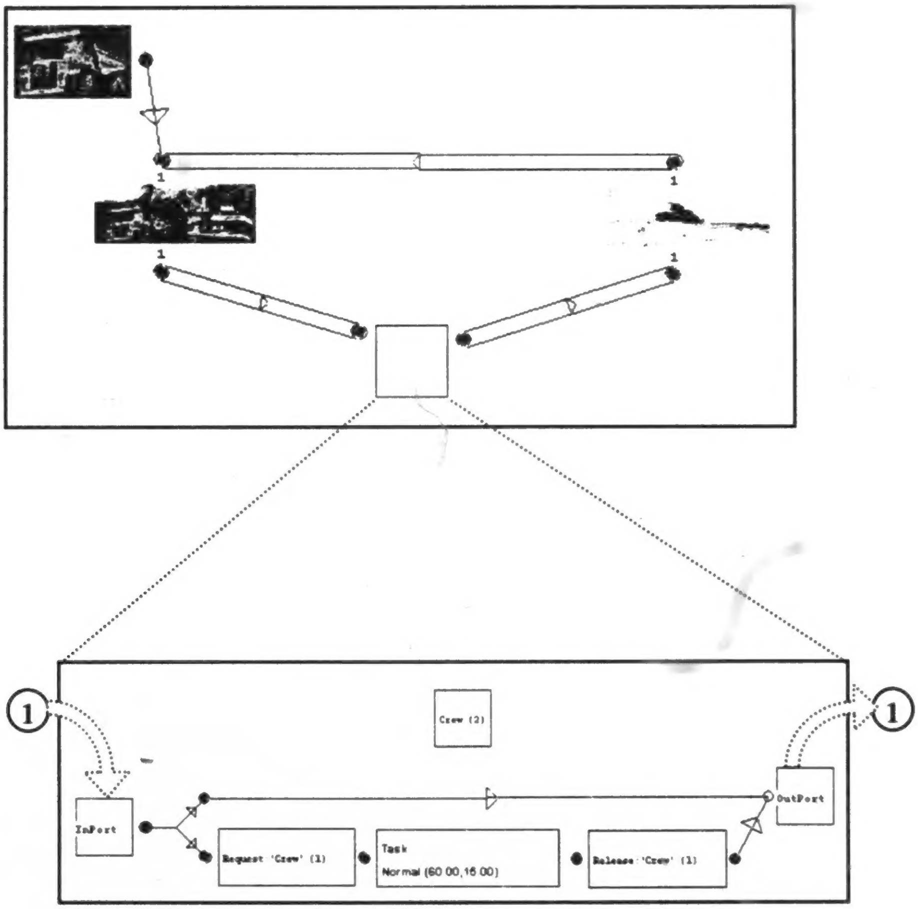

1
1
1
1
①
①
Crew (2)
InPort
OutPort
Request 'Crew' (1)
Task
Normal (60.00,15.00)
Release : 'Crew' (1)

Figure 7-9 Extending a SPS Tool with the Help of Generic Modeling Elements

7.6 Issues of Tool Development and Utilization

7.6.1 Introduction

In the beginning, there were general-purpose simulators, which were difficult to use. In construction research's endeavor to increase the user base of simulation, simpler modeling constructs have been created. However, simplification of those modeling constructs resulted in their inability to model complicated situations. As a result, the simple constructs were extended to give them more functionality and new constructs were added. Further, some systems allowed for the definition of construct behavior in a loose form such as a program script with dynamic capabilities. Ironically this trend represents a cycle of development that leads back to full-blown general-purpose simulation. The argument is that a user who is knowledgeable in the new and enhanced environments should also be able to easily learn the general-purpose simulation environment. The problem is that the development of simulation systems is a time consuming and difficult task involving expert programmers who possess knowledge of object-oriented development, database programming, and graphical user interface development while also possessing an understanding of the construction process itself.

A hybrid concept was developed that redefined the roles and responsibilities of both the tool developer and the tool user. This concept accommodates both novice and expert users. Further, it allows for a flexible environment that allows users to do their own kind of tool extension or development. The separation between the two groups becomes fuzzy as developers utilize a simplified development environment while users have some development capability. A symbolic representation of the various users and their levels is illustrated in Figure 7-10.

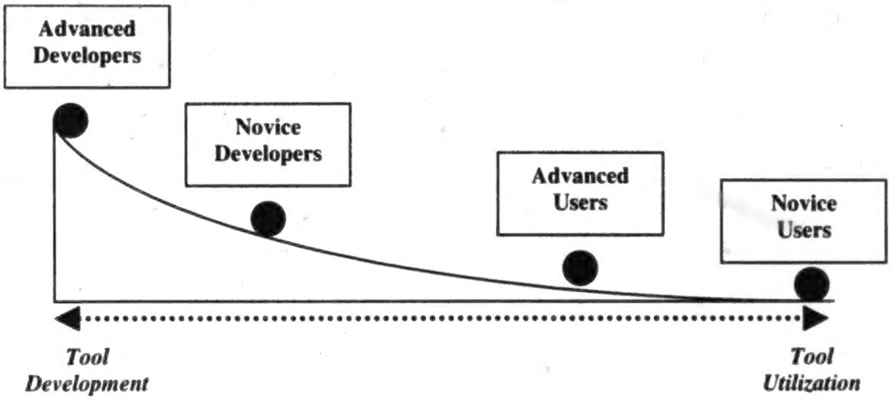


Figure 7-10 Accommodation of Users and Developers with Varying Degrees of Skill

The idea is to provide a development environment that simplifies the process of creating new tools and a modeling environment that allows and even encourages the use of development type features.

7.6.2 Simplified Development

The development environment for new SPS tools is simplified as follows:

- A code library that encapsulates all the routines commonly required by simulation tools is used. This includes discrete event simulation, graphical drawing, statistical collection, tracing, and database access routines. This library is designed as an application framework making it flexible and extendable. Application frameworks are discussed in more detail in Chapter 10. By building this library, development efforts that are not directly related to modeling are reduced and developers can focus on optimizing the core modeling and simulation related code. Further, the library becomes a sort of standard that is shared by the developed tools.

- New elements are defined by customizing a supplied generic modeling element that comes with a wide range of default functionality. More information on the structure of such an element is provided in the next section. By utilizing a common base element, compatibility between developed elements, including those from different tools, is an implicit process.
- The development environment itself is controlled and customized, allowing developers to focus on providing the required information rather than getting lost in the host of displayed features. This is demonstrated in Chapter 9.
- By broadening the definition of a developer through the streamlining of the development environment, small and large companies can now afford their own simulation tool development. This should lead to increased use of simulation for project planning. Users who feel that the existing tools are limited or who believe that a new tool is worth developing for a given domain can contact their local information system group to get this done in a reasonable amount of time.

7.6.3 Flexible Modeling Environment

With the described approach, users are also treated as special developers; they are given the ability to extend existing elements or even create new ones. This is done as part of an advanced modeling environment by allowing users to write code inserts in the form of programming statements in a manner similar to general-purpose simulation languages. The language of extension is the same as the language originally used to develop the elements. This means that users are in fact acquiring the knowledge to be developers.

In order to create new modeling elements as part of the modeling environment itself, modular and hierarchical modeling concepts are used in association with a set of generic set of modeling elements and a user element library. This was illustrated in Section 7.5 .

7.7 Object-oriented Modeling

Object-oriented modeling is based on the idea of object-oriented programming. It is mentioned in this chapter as opposed to the system and implementation chapters because it constitutes a critical part of the methodology itself. It is an enabling technology that makes the goal of unifying all other concepts presented in this chapter reachable.

An object-oriented approach can be used for the representation of all structures of a simulation environment, including the code library and the project information. The result is an object library, which provides the means of accessing and manipulating simulation projects. A graphical user interface simply provides an interface between users and the library. This opens up the possibility of bypassing the GUI to perform batch operations or to extract simulation data for use by other systems. For example, assume that a given model requires the creation of fifty connection road segment elements. Using the GUI, the user would have to create each element individually and ensure they are placed and connected appropriately, a process which could take several hours. Alternatively, users can choose to temporarily bypass the GUI and construct a script to perform this task. This script might appear as follows:

```
Dim I as integer
For I = 1 to 50
        AddElement "RoadSegment", 100, I * 80
Next
For I = 2 to 50
        AddRelation Elements(I), Elements(I-1)
Next
```

7.8 Unified Modeling Methodology

7.8.1 Overview of Unification approach

It is proposed that the presented concepts be combined into a unified modeling methodology through a generic base modeling element. This generic element would not be directly used for modeling purposes. Rather, it would be used as the starting point for specific modeling element developments. Implementation of new tools becomes a simplified process where developers create the set of required modeling elements based on this generic element. Another requirement is that the development process be controlled, meaning that developers need only define what actions to take in response to certain state changes affecting the respective modeling element. To further simplify the development process, a toolkit development approach is advocated whereby several libraries, called services, would be made available to developers for supporting common tasks such as discrete event simulation processing and statistical analysis.

7.8.2 Generic Base Modeling Element

Using object-oriented language, the generic base modeling element is a virtual base class that encapsulates data structures representing a host of information, including connection points, parameters, outputs, results, and simulation resources. Class methods provide the means to manipulate these structures and to perform other operations. Class events provide feedback as to changes in the state of an element. Class properties, methods and events that are related are grouped together as part of a class behavior. It is through these behaviors that the concepts that make up the methodology take shape. Figure 7-11 presents the structure of the proposed generic base modeling element in terms of its behaviors. Detailed information on the supported behaviors and their related properties, methods, and events will be discussed in the next chapter (Section 8.2.2).

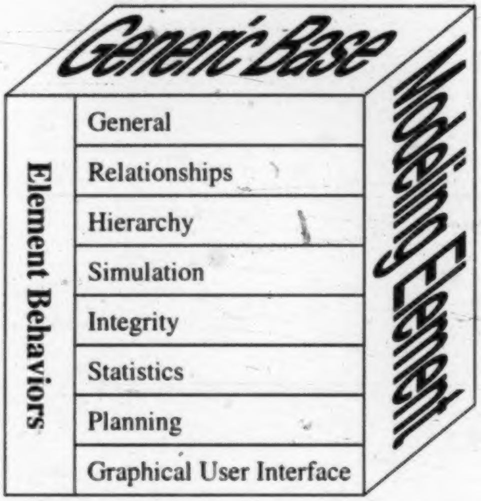

Generic Base
Modeling Element
Element Behaviors
General
Relationships
Hierarchy
Simulation
Integrity
Statistics
Planning
Graphical User Interface

Figure 7-11 Behaviors of the Generic Base Modeling Element

7.8.3 SPS Tool Development

The development process for new SPS tools would involve the creation of the required modeling elements and their customization. Customization of new elements would be done by providing the set of actions, in the form of computer code, to be taken in response to certain events. The generic base modeling element would also include intelligence in the form of default implementations for most events. This would allow developers to concentrate on the uniqueness of each element.

This proposed approach to tool development constrains developers to the provided architecture of the system. This means that they can't simply add a new behavior.

However, this architecture, through the generic base modeling element, becomes a kind of modeling standard to which all new tools must conform.

7.8.4 The Support Libraries

Developers should also be able to access a set of pre-defined services during the customization of new modeling elements. Services are code libraries that encapsulate a set of commonly used routines for such tasks as discrete event simulation processing, statistical analysis, tracing, and database access. The unified modeling methodology stipulates that a set of support libraries must exist that allow developers to focus on coding the core requirements of their tools. The actual transformation of these requirements into actions is the responsibility of the respective service. The identified set of libraries, also called services, required for SPS tool development and their relationships to the modeling element behaviors are illustrated in Figure 7-12. Detailed information on the proposed services and the functionality they expose will be provided in the next chapter (Section 8.2.3).

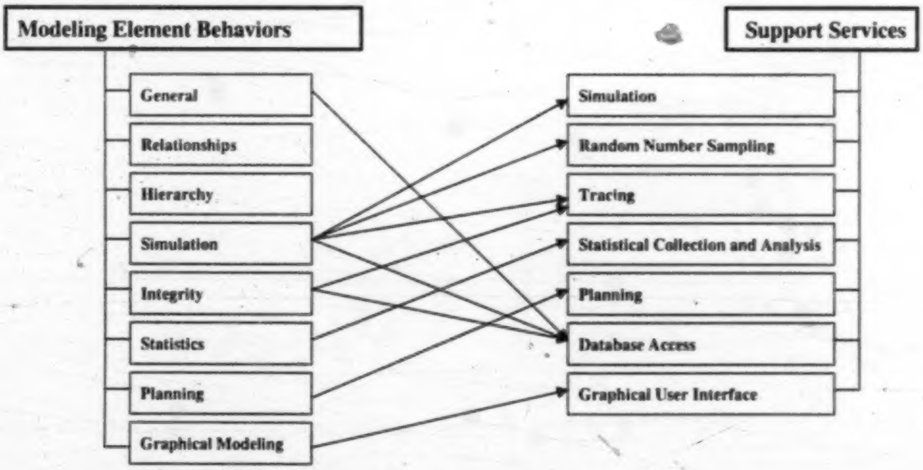

Modeling Element Behaviors
General
Relationships
Hierarchy
Simulation
Integrity
Statistics
Planning
Graphical Modeling
Support Services
Simulation
Random Number Sampling
Tracing
Statistical Collection and Analysis
Planning
Database Access
Graphical User Interface

Figure 7-12 List of Identified Services and Relationships to Element Behaviors

7.8.5 Simulation Modeling

A simulation model becomes a collection of instances of modeling elements. Figure 7-13 illustrates this definition. Modeling element instances of the same type (i.e. based on the same modeling element) would share the same code and be differentiated solely by the value of their properties.

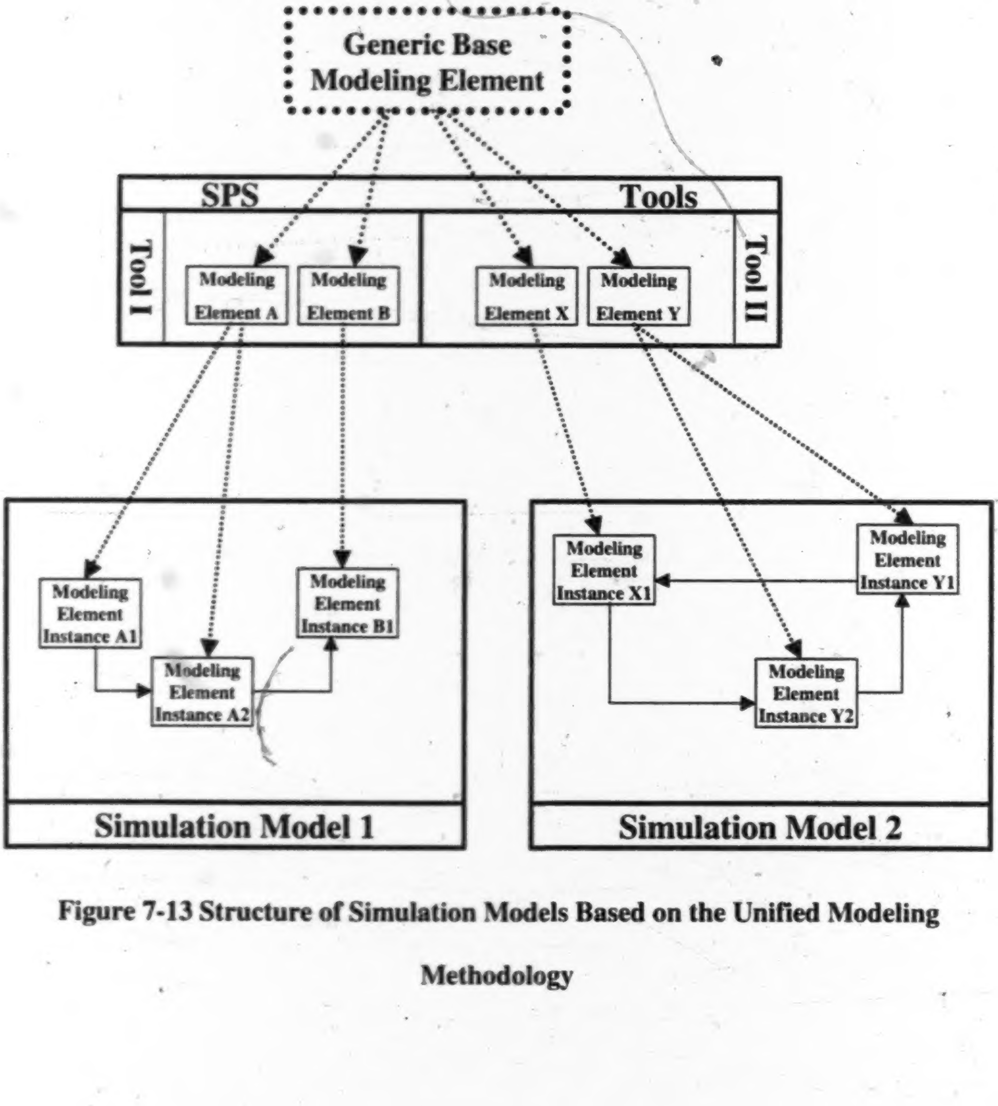

Generic Base Modeling Element
SPS
Tools
Tool I
Tool II
Modeling Element A
Modeling Element B
Modeling Element X
Modeling Element Y
Modeling Element Instance A1
Modeling Element Instance A2
Modeling Element Instance B1
Modeling Element Instance X1
Modeling Element Instance Y1
Modeling Element Instance Y2
Simulation Model 1
Simulation Model 2

Figure 7-13 Structure of Simulation Models Based on the Unified Modeling Methodology

7.9 Conclusions

The proposed unified modeling methodology presented in this chapter is based on several developed concepts. Special purpose simulation modeling allowed for the development of intuitive domain-based modeling elements. Graphical modeling allowed for the simplification of the modeling process for novice users. Integrated modeling both defined how information can be obtained from other systems through a relational database management system and introduced a standard for the generation of project planning type of data. Modular and hierarchical modeling allowed for the modeling of large simulation projects, linking of models based on different SPS templates, and empowerment of users with development capabilities. The hybrid tool development and utilization approach explained how it is more beneficial for a simulation system to support a spectrum of developers and users with different skills and how their needs can be satisfied. Finally, object-oriented modeling was introduced as an enabling concept that can bridge the gap between all the presented concepts and also allow for direct access to the simulation projects through an exposed object library.

It is proposed that the unification of these concepts can be accomplished through a generic base modeling element, which can be customized by developers for the creation of specialized modeling elements. The development process would follow the toolkit approach where several services support developers during the customization process.

Chapter 8 – Simphony Environment

8.1 Introduction

The methodology presented in the previous chapter was used to develop a construction simulation system called Simphony. This chapter will provide a detailed description of the various components of this system. A high level depiction of the overall environment is presented in Figure 8-1.

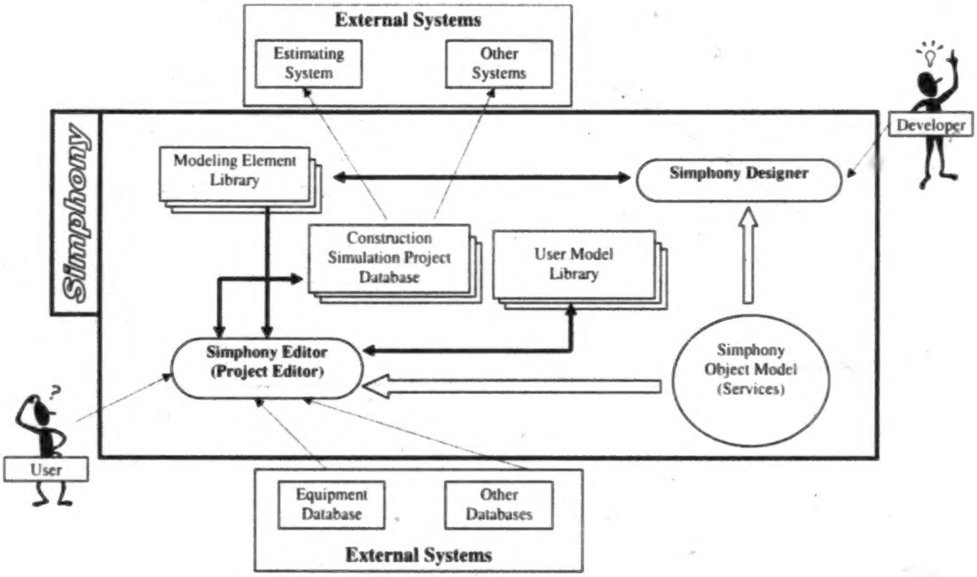

External Systems
Estimating System
Other Systems
Simphony
Modeling Element Library
Simphony Designer
Developer
Construction Simulation Project Database
User Model Library
Simphony Editor (Project Editor)
Simphony Object Model (Services)
User
Equipment Database
Other Databases
External Systems

Figure 8-1 Simphony Environment

Developers utilize the Simphony Editor program to create SPS tools called "SPS Templates". This component was referred to as the "Project Editor" in the previous chapter. SPS templates are a collection of modeling elements targeted for a single domain. The term "Developer" will be used in the rest of this chapter to denote any person engaged in this process. Developed templates are stored in the Modeling Element

Library. Users utilize the Simphony Editor to create simulation models based on the existing SPS templates in the Modeling Element Library. The term "User" will be used throughout this chapter to refer to users whose primary role is building simulation models. Constructed simulation models and their simulated results are stored in and retrieved from the Construction Simulation Project Database. This is a relational database management system that other systems can access to extract the desired information. The User Model Library is another such database that is utilized by users to store and retrieve reusable simulation models.

8.2 SPS Template Development and the Simphony Designer

8.2.1 Introduction

Creating new SPS templates involves the design and implementation of the modeling elements that will be used to create simulation models for a given domain. The design component is not a trivial task. It involves a complete understanding of the targeted construction domain itself and the fundamental principles of SPS modeling. Section 8.2.5 will present a formal procedure that can be followed during the design stage.

Once the design is complete, implementing the template involves the creation of the required modeling elements through the Simphony Designer. Creating new modeling elements involves the customization of the behaviors of a generic base modeling element. This, in turn, involves writing code in the form of event handlers in response to the various events. Simphony generates these events in response to user or system actions. Defining a given behavior requires complete understanding of the associated data structures, methods and events. Section 8.2.2 will detail the structure of the generic modeling element and its behaviors. Section 8.2.6 will present a detailed guideline that developers can follow during the implementation stage.

A library is available to support the developers during the implementation process. This library provides a wide range of commonly used routines including discrete event scheduling, statistical collection and analysis, simulation tracing, and graphical user interface support. These routines are grouped by their function into services. Information on the supported Simphony services is discussed in detail in Section 8.2.3.

Simphony Designer is the computer program used in the implementation of new templates. It is similar to a typical integrated development interface in that it combines the processes of code entry, validation and compilation into a single program. Access to Simphony services is also made available through this program. Simphony Designer is presented in detail in Section 8.2.4.

To demonstrate the versatility and flexibility of the described system, a template of generic modeling elements was developed to allow for the use of general purpose modeling concepts. The development of this template along with code samples will be provided in Section 8.2.7.

8.2.2 Generic Modeling Element

8.2.2.1 Introduction

The generic modeling element is used by developers as a starting point for the creation of new modeling elements. Customization of this generic element involves the definition of its various behaviors by implementing event handlers and manipulating the specific instances using properties and methods.

Table 8-1 provides a complete list of the supported modeling element behaviors and their related properties, methods and events. The remaining subsections will give an overview of each behavior and the means by which a developer can customize their implementation. These sections will also begin to introduce the coding syntax developers are expected to provide.

Newly developed modeling elements are added to the modeling element library. Once in the library, the user can use them to create multiple instances as part of the modeling process. Each instance uses a single set of code. Instances based on the same modeling element differ according to the values of their properties.

Table 8-1 Modeling Element Behaviors and Associated Properties, Methods and Events

Behavior	Properties	Methods	Events
General	ID ElementType Attr	Delete AddAttribute	OnCreate OnDelete
Relationships	ConnectionPoints	AddConnectionPoint DeleteConnectionPoint AddRelation DeleteRelation	OnAddRelationIn OnAddRelationOut OnDelRelationIn OnDelRelationOut
Hierarchy	ChildElements Parent	AddElement DeleteChildren	OnAddElement OnDeleteElement
Simulation	CurrentEntity File FirstEvent Res	AddEntity AddEvent AddFile AddResource CancelEvent CloneEntity DeleteEntity PreemptResource RequestResource ReleaseResource ScheduleEvent StopSimulation TransferOut	OnSimulationInitialize OnSimulationInitializeRun OnSimulationPostRun OnSimulationProcessEvent OnSimulationTransferIn

Integrity		AddAttribute	OnCheckIntegrity OnRelationValid OnValidateParameters
Statistics	Stat		
Planning	Activity	AddActivity AddStatistic	
Graphical Modeling Support	CoordinatesX CoordinatesY Selected	SetNumAttributes GetNumCoordinates AddAttribute DrawConnectionPoints	OnDraw OnDragDraw OnDrawRelation OnGetBoundingBox OnHitTest OnListBoxInitialize OnListBoxSelectItem OnMove

8.2.2.2 General

Several properties, methods and events are available for general use and are not specific to any particular behavior.

When the user attempts to create an instance based on a given modeling element, the OnCreate event is triggered. This is similar to the concept of a class constructor in an object-oriented language. This event allows the developer to define the initial structure of the element and declare any required user attributes, files, resources or statistics. The OnDelete event is triggered when the user deletes an existing element instance. Developers can handle this event to perform some functions prior to the deletion of the instance.

The Delete method is used to explicitly delete a given element instance; the AddAttribute method is used to declare user attributes. User attributes are used to provide parameters that users can modify during the modeling process. Attributes can also be used to provide read-only information and to hold temporary internal values. Developers can declare as many attributes as needed for each modeling element. Attributes are the only means for the representation of state information of a given modeling element instance. This is important because it allows for automatic object persistence. This means that a given simulation project can be automatically saved and restored by simply saving and restoring the attribute values of all the defined modeling element instances.

The AddAttribute method takes a number of parameters that determine the internal and external representation, allowable user access type, and valid numeric range for numeric type attributes. The internal representation (i.e. data type) of an attribute can be numeric, date, time, textual, array, distribution, or object. The external representation and the user access type are related to the graphical user interface representation and are discussed in Section 8.2.2.9.5. The allowable range parameters are related to the integrity behavior and are discussed in section 8.2.2.6.3.

The ID property is a internally set numeric value that uniquely identifies a given modeling element instance. The ElementType property returns a textual value that indicates the name of the base modeling element of the instance. The Attr property is used to access the value of a previously declared attribute.

8.2.2.3 Relationships

Relationships between modeling element instances are supported with the notion of a connection point. A connection point is a data structure that is capable of holding a collection of references to connection points in other elements. Connection points for a given modeling element are declared with the AddConnectionPoint method, which is done as part of the OnCreate event handler. This method expects a number of parameters that determine the type, initial geometrical coordinates and tolerance of the connection point. The type can either be input or output. The remaining method parameters are related to the graphical modeling behavior and will be discussed in Section 8.2.2.9.4. The AddRelation and the DelRelation methods can be used for the manual manipulation of relationship information.

When the user manipulates relationships between two element instances through the graphical user interface or through code, several events are triggered to inform the developer of changes in the relationship information. The OnAddRelationIn and OnAddRelationOut events are triggered when the user requests the addition of a relation between two connection points. The former is triggered for the destination instance while the latter is triggered for the source instance. Similarly, the OnDeleteRelationIn and the OnDeleteRelationOut are triggered after a relationship is deleted. This process is illustrated in Figure 8-2.

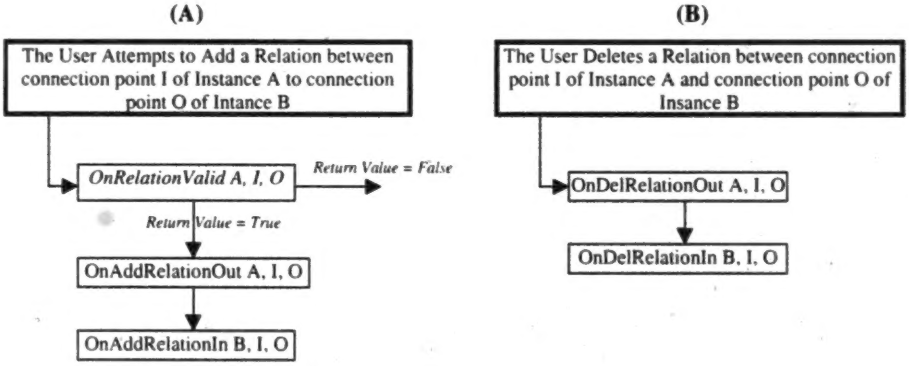

(A)
(B)
The User Attempts to Add a Relation between connection point I of Instance A to connection point O of Intance B
OnRelationValid A, I, O
Return Value = False
Return Value = True
OnAddRelationOut A, I, O
OnAddRelationIn B, I, O
The User Deletes a Relation between connection point I of Instance A and connection point O of Insance B
OnDelRelationOut A, I, O
OnDelRelationIn B, I, O

Figure 8-2 Triggered Events in Response to User Manipulation of Relationships

The shown OnRelationValid is related to the integrity behavior and will be discussed in more details in section 8.2.2.6.2.

8.2.2.4 Hierarchy

With hierarchical modeling, certain modeling element instances can be created as children of other elements. The parent instance may contain higher-level simulation model resources and attributes and is mainly responsible for routing incoming entities to the appropriate child elements.

The ChildElements property of the modeling element allows developers to navigate the list of child elements defined while the Parent property can be used to access the parent element. The AddElement method is used to create a child element instance while the DeleteChildren method deletes all the child element instances.

The two events specifically related to hierarchical behavior are OnAddElement and OnDeleteElement. The OnAddElement event is triggered for the parent of the element instance just added to a given model, while the OnDeleteElement is triggered for the parent instance of an element about to be deleted. The complete event sequence diagram for element addition and deletion is shown in Figure 8-3.

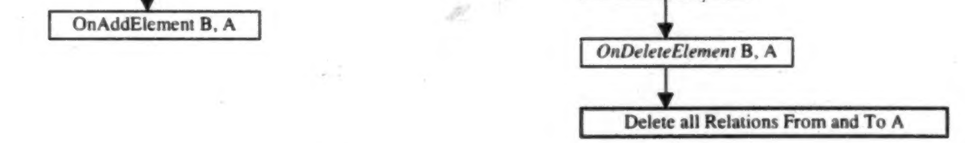

(A)
(B)
Element A is added as a child of Element B
OnCreate A
Return Value = False
Return Value = True
OnAddElement B, A
Element A who is a child of element B is Deleted
Delete All Child Elements of Element A
OnDelete A
OnDeleteElement B, A
Delete all Relations From and To A

Figure 8-3 Events Triggered in Response to the Addition and Deletion of Elements

8.2.2.5 Simulation

8.2.2.5.1 Overview

Definition of the simulation behavior of a given element involves (1) the declaration of the required resources, files, and events, (2) the initialization of the starting entities and the scheduling of the initial simulation events, (3) the processing of entities and events, and collection of statistics, and (4) the final analysis of the results.

The four basic elements of a discrete event simulation model are entities, events, resources and files. Entities represent active elements of the model and can have any number of attributes (eg: Truck Capacity, Truck speed, ...). A simulation event represents a transition in the state of the simulation. Resources are used to model limited resource capacity where entities require those resources for given tasks. Files represent queues of entities waiting for services due to unavailability of resources.

8.2.2.5.2 Resources

Resources can be used to simplify the process of implementing a simple multi-server situation tied to a single queue. They are usefull for situations where incoming entities should first wait in a queue for a resource to be available and then proceed when possible. Resources are first declared with the AddResource method in the implementation of the OnCreate event handler. This means that they must be defined even before the simulation starts. However, the number of resources for a given resource type can be changed during the simulation initialization. Once declared, the created resource object can be obtained with the Res member property. During simulation processing, resources are requested with the RequestResource member method and released with the ReleaseResource member method.

8.2.2.5.3 Files

Files can be used to represent a queueing situation that cannot be modeled with a resource. This includes situtations where multiple queues are tied to a single resource or custom resource allocation schemes are required. Figure 8-4 illustrates such a situation where incoming trucks queue up according to their size. Files are also first declared in the OnCreate event handler with the AddFile member method and then accessed with the File modeling element member property.

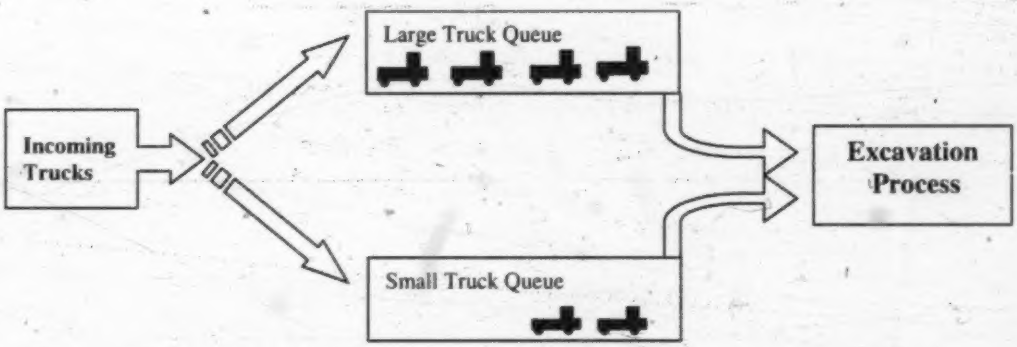


Figure 8-4 Utilization of Files to Represent a Truck Queuing Situation

8.2.2.5.4 Simulation Events

Simulation events allow for discrete event simulation to take place. Simulation events are first declared in the OnSimulationInitialize event handler. Simulation events should not be confused with modeling element events. Modeling element events like OnCreate, and OnSimulationInitialize provide means of customzing the modeling element behavior. Simulation events can be scheduled to occur at a certain time through the ScheduleEvent member function and then processed as part of the implementation of the OnSimulationProcessEvent modeling element event.

8.2.2.5.5 Entities

Entities, which are also fundemental to the implementation of discrete event simulation, are created with the AddEntity member method. Entities are usually assigned attributes that can be used to make certain decisions during the simulation. Some modeling elements will create entities, typically in the OnSimulationInitializeRun event handler, and others will simply process entities as they are transferred from other elements.

8.2.2.5.6 Inter-Element Communication

Communication between element instances can be done either by one element directly manipulating another element's attributes, or, in most cases, through entity transfers. The transfer process is automated according to the declared information. When an entity needs to be transferred from a given element through one of its connection points, a call is made to the TransferOut method. The first argument to this method is the entity to be transferred out. For each destination element instance connected to any of the connection points (i.e. which has a relation with the current element instance), the entity will be cloned and the destination instance's OnSimulationTransferIn event will be triggered with the appropriate parameters. The default implementation of the OnSimulationTransferIn event is to schedule the designated "FirstEvent" of the element. The "FirstEvent" is assigned when the simulation events are created with the AddEvent method. This approach accomplishes the modularity concept previously described in

Figure 7-6. Customization of this default mechanism can also be performed through the custom implementation of the OnSimulationTransferIn event.

8.2.2.5.7 Event Sequence for Simulation Behavior

The diagram shown in Figure 8-5 illustrates the complete list of events that are triggered when the user requests that simulation be initiated.

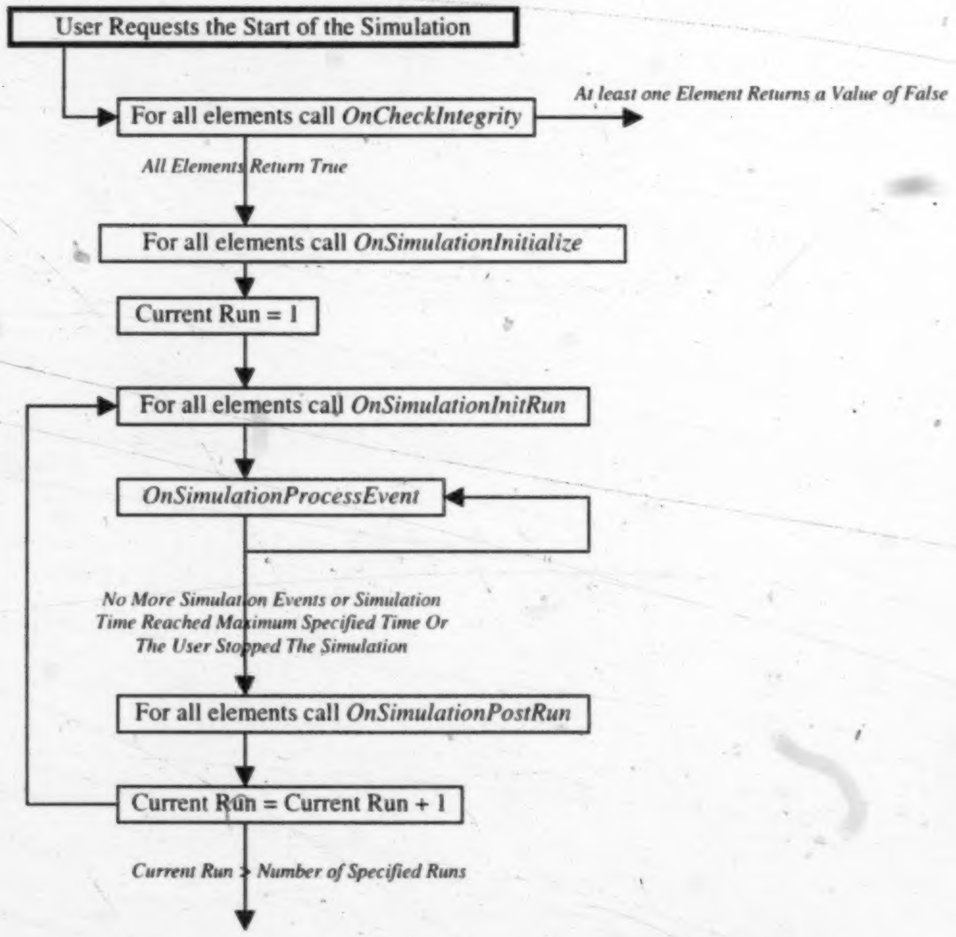

User Requests the Start of the Simulation
For all elements call *OnCheckIntegrity*
At least one Element Returns a Value of False
All Elements Return True
For all elements call *OnSimulationInitialize*
Current Run = 1
For all elements call *OnSimulationInitRun*
OnSimulationProcessEvent
No More Simulation Events or Simulation Time Reached Maximum Specified Time Or The User Stopped The Simulation
For all elements call *OnSimulationPostRun*
Current Run = Current Run + 1
Current Run > Number of Specified Runs

Figure 8-5 Triggered Event Sequence for Simulation Behavior

The OnCheckIntegrity is related to the integrity behavior and will be discussed in Section 8.2.2.6.4. The OnSimulationInitialize event follows an OnCheckIntegrity event. Some of the tasks typically performed in this event handler are to:

- Reset the number of resources for resources based on the values of certain element attributes. This is the only event where changes in the number of resources for resources are possible.
- Declare the simulation events using the AddEvent method.
- Initialize project planning information.

The OnSimulationInitializeRun event is called at the beginning of each run. In this event, the starting simulation entities are created and scheduled with their starting events. The OnSimulationPostRun is called at the end of each run. The most significant event is OnSimulationProcessEvent, which is called whenever a previously scheduled simulation event should be processed. The OnSimulationPostRun allows for any post simulation run analysis or cleanup.

8.2.2.6 Integrity

Defining the integrity behavior involves the enforcement of certain rules at different stages during the modeling process. Integrity rules can be grouped into four categories, which are described in the next four subsections.

8.2.2.6.1 Genealogy-Based Rules

Genealogy-based rules are designed to prevent invalid parent-child type relationships where certain child elements cannot exist outside the scope of a given parent element. This dependency could be due to the fact that the child requires certain attributes or simulation resources of the parent. Support for this type of rule is provided through the OnCreate event handler. Referring back to Figure 8-3A, when the creation of a new element instance is requested, the OnCreate event is triggered. If this event returns a value of "False", then the instance will not be created. This gives developers a chance to check the type of the parent element through the Parent and the ElementType properties and perform the required validation.

8.2.2.6.2 Relationship-Based Rules

In certain situations, the simulation model logic places certain constraints on the predecessor and successor relationships of element instances. This includes such rules as "Element Type A cannot precede Element Type B" or "Element Type A cannot participate in more than one relation". Support for these types of rules is provided through the OnRelationValid event, which is triggered when the user requests that a relation be added between two modeling elements (see Figure 8-2A). If the return value from the associated event handler is "False" then the relation will not be added. Developers can use this event to perform the required validation.

8.2.2.6.3 Parameter-Based Rules

Parameter type attributes allow users to modify their values and affect the outcome of the simulation. Validation rules on the supplied user data are supported by the OnValidateParameters event.

When the attribute is initially declared with the AddAttribute method, developers have the capability to provide a minimum and a maximum value as method parameters. These supplied values are used in the default implementation of the OnValidateParameters event which ensures that users are not allowed to supply values outside the valid range. This obviously only applies to numeric types of attributes. For extended parameter validation requirements based on complex numeric rules, text validation rules, or even rules involving more than one attribute, developers can provide their own implementation of the OnValidateParametes event.

8.2.2.6.4 Model Based Rules

All the three previous types of integrity rules trap potential errors before they occur. This is the desired mean of controlling model integrity but it is not always possible. SPS modeling, in association with graphical modeling, utilizes a modeling approach that is non-linear in nature. As a result, the simulation model might contain intermediate states which are invalid. For this reason, model-based rules are required. Examples of such rules are "Element A must contain at least one child instance of Element B" or "Element A must have a relation with another instance of type Element B". Such rules are supported with the OnCheckIntegrity event, which is triggered when the user requests that simulation be iniated (see Figure 8-5). If a single modeling element returns "False" for this event, then the simulation will not be started.

8.2.2.7 Statistics

Statistical collection behavior for a given modeling element is enabled by declaring the statistic in the OnCreate event handler with the AddStatistic member method. Declared statistics can be accessed through the Stat member property. The actual collection and analysis of the data is the responsibility of the statistical collection and analysis service; this will be discussed later in Section 8.2.3.5.

Explicit declaration of statistics is only requried for custom statistics. The declaration of Resources whithin a given element automatically implies the declaration of three default statistics: Utilization, WaitingTime, and Queue length. Similary, the declaration of Files automaticaly implies the declaration of two statistics: Waiting Time and Queue Length. Statistical analysis results are displayed as part of the element attribute dialog form. This is the responsibility of the graphical user interface services (see Section 8.2.3.8.4).

8.2.2.8 Planning

Modeling elements which need to generate project planning information must enable the planning behavior. This involves an initial call to the AddActivity method, as part of the OnCreate handler, to declare the project activities. Once an activity is created, it can be accessed with the Activity property. The actual generation of planning data, including resource, cost, production and revenue information, is done through the planning service; this will be discussed in Section 8.2.3.6.

8.2.2.9 Graphical Modeling Support

8.2.2.9.1 Overview

Definition of the graphical modeling behavior for a given element involves initializing certain settings and responding to certain events generated by the GUI. The actual rendering of graphical primitives, dialog boxes and forms is the responsibility of the GUI service; this is discussed in detail in Section 8.2.3.8.

8.2.2.9.2 Geometry

Graphical modeling requires that modeling elements support advanced two-dimensional graphical representation and manipulation. This implies that each element will require an associated set of geometrical coordinates that determine its form on a model layout window.

To support this requirement, two array type properties, CoordinatesX and CoordinatesY, are available. The size of the arrays can be accessed with the SetNumCoordinates and the GetNumCoordinates methods. Most modeling elements will only require one set of coordinates representing the top left corner of the graphical representation with a constant width and height. Other elements, which might require a dynamic shape that can change as a result of certain user or internal actions, employ multiple coordinates. Figure 8-6A demonstrates a simple element that utilizes a single set of coordinates while Figure 8-6B demonstrates how two sets of coordinates are employed for an element that requires flexible representation.

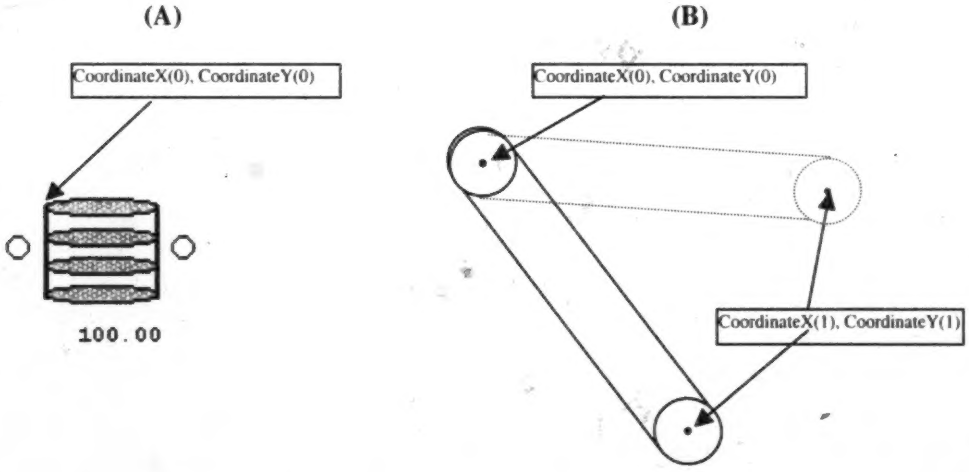

(A)
(B)
CoordinateX(0), CoordinateY(0)
100.00
CoordinateX(0), CoordinateY(0)
CoordinateX(1), CoordinateY(1)

Figure 8-6 Use of Geometrical Attributes as Modeling Element Reference Points

Once the geometry of the modeling element is defined and initialized, the default implementation of geometry manipulation tasks is automated through the default implementation of the relevant events. However, if they wish, developers can still implement their own custom implementation by defining their own event handlers.

When the user attempts to modify the geometry of a given element instance through drag and drop mouse operations, the OnMove event is triggered. The default implementation offsets all the coordinates and the coordinates of the connection points by the amount of movement that the user performs. If custom handling of dragging operations is required, (for example, to implement rotation) then the developer defines his own OnMove event handler.

8.2.2.9.3 Element Representation

Graphical element representation is supported through the OnDraw event, which is triggered when the screen display is refreshed. Developers can utilize a wide range of graphical display methods, which are available through the graphical user interface service (see Section 8.2.3.8.2).

8.2.2.9.4 Relationship Representation

The OnDrawRelation event is triggered when a graphical representation of the relationships between two connection points is required. The default implementation draws an arrow between the two connection points.

Connection points also need to be represented graphically as part of the OnDraw event handler. The default representation is available through the member method

DrawConnectionPoints and is a circle. Developers can choose not to use this method and implement their own connection point representation instead.

The actual manipulation of relationship information between elements can also be done through the GUI. Users can add or delete relations by selecting the source and destination connection points. This is called explicit manipulation of relationships. Another supported method of creating relationships, denoted implicit manipulation, automatically adds a relation between two connection points if the distance between their coordinates is smaller than the tolerance value supplied by the developer when the connection point was originally declared.

8.2.2.9.5 Graphical Attribute Representation

Developers can define the type of GUI controls and access permissions that users have for manipulating exposed element attributes. When the attributes are initially declared with the AddAttribute method, one of the method parameters represents the type of external representation for the attribute. Possible options include text box, calendar, distribution selector, list box, table, and graph.

Another AddAttribute parameter is also available to control the type of access that users have over the attribute value. Possible options are (1) "Read-Write", indicating the attribute is a parameter that can be changed by users, (2) "Read-Only", indicating an output type attribute that users can view but not change and (3) "Hidden", indicating that the attribute is for internal usage only.

8.2.3 Simphony Services

8.2.3.1 Introduction

Simphony services encapsulate commonly used routines, which can be used directly or indirectly by the developer during the implementation of the event handlers. This section will present the available services and the support they provide.

Access to these services during the development process is transparent to the developer. Some services are available explicitly through special statements while others are implicitly accessed through member methods and through member properties of the modeling element class. Table 8-2 summarizes the means of accessing the supported services and the routines they expose.

Table 8-2 List of Simphony Services, their Access points, Properties and Methods

Service	Access Point	Properties	Methods
Simulation	Modeling Element	CurrentEntity FirstEvent File Resource	AddEntity, AddEvent, AddFile, AddResource, CancelEvent, CloneEntity DeleteEntity, PreemptResource, RequestResource, ReleaseResource, ScheduleEvent StopSimulation,
Random Number Sampling	Sampler statement Or Distribution type Attributes	DistType, NumParameters, ParamValues, Positive, Stream	Seed, Beta, Expntl, Normal, Triang, Uniform Value
Tracing	Tracer statement	LastMessage TraceEnabled	Trace
Statistical Collection and Analysis	Modeling Element (Stat property)	Intrinsic, Average, MaxVal, MinVal GlobalAverage GlobalStdDev	Collect, SetFinish, SetStart

		GlobalMin, GlobalMax	
Planning	Modeling Element (Activity property)		AddCostCenter AddResource, Halt, Start, OffsetResourceLevel RecognizeCost RecognizeProduction RecognizeRevenue
Database access	dbengine statement	See section 8.2.3.7	
Graphical User Interface	CDC statement		Arrow, ArrowHead, Ellipse, Circ LineTo, MoveTo, Rectangle, RenderPicture, TextOut, ChangeFillColor, ChangeFont, ChangeLineStyle, ChangeTextColor, TextHeight TextWidth

8.2.3.2 Simulation

The simulation service encapsulates a discrete event simulation engine that is responsible for managing all entities, resources, files and scheduled events. Access to this service is provided through the previously discussed simulation behavior methods of the modeling element (see Section 8.2.2.5). Figure 8-7 illustrates the components of the simulation service and its access points with the modeling element methods.

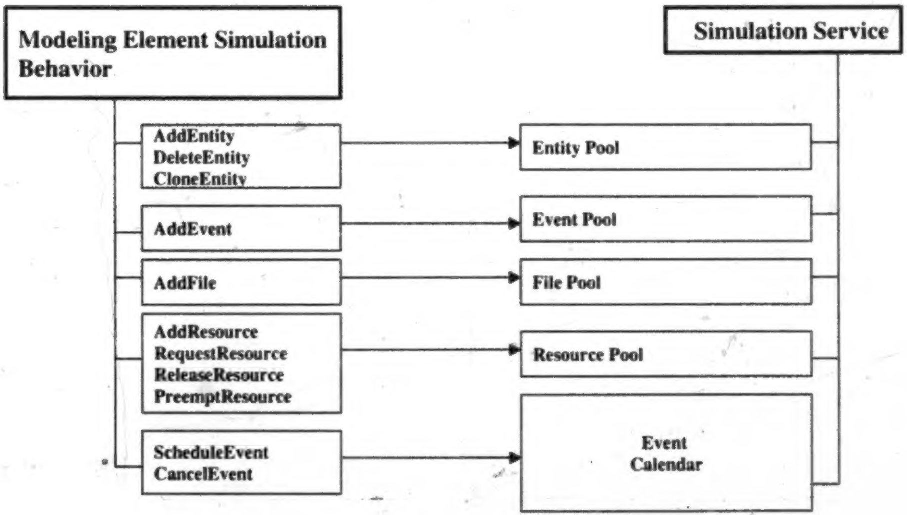

Modeling Element Simulation Behavior
Simulation Service
AddEntity
DeleteEntity
CloneEntity
Entity Pool
AddEvent
Event Pool
AddFile
File Pool
AddResource
RequestResource
ReleaseResource
PreemptResource
Resource Pool
ScheduleEvent
CancelEvent
Event Calendar

Figure 8-7 Simulation Service Components and its Access Points

The entity pool manages the set of defined entities in the model and allows modeling elements to add, delete or clone entities. The event pool manages the list of events defined with the AddEvent method. The file pool encapsulates the list of defined files and provides the means to navigate and manipulate their content. The resource pool manages the set of defined resources and handles all requests for resource allocations and releases.

The event calendar is responsible for the management of the simulation events. Simulation events are added with the ScheduleEvent method. Their position within the internal data structure is dependent on the event time supplied through the ScheduleEvent method parameters. Scheduled events can also be cancelled with the CancelEvent method. Simulation processing is initiated when the user requests the start of the simulation. A representation of the underlying algorithm responsible for this process is shown in Figure 8-8. The simulation event loop first removes the next event to be processed from the event calendar and advances the current simulation time to the time of the event. The event is then routed to the modeling element instance that originally scheduled it. During simulation processing, instances may schedule further events, which are in turn inserted into the appropriate calendar position. This process continues until all the events in the calendar have been processed or simulation time reaches a maximum specified by the user.

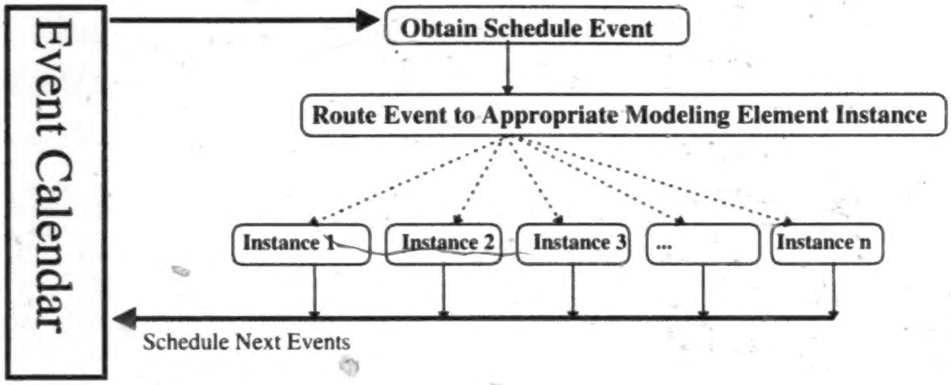

Event Calendar
Obtain Schedule Event
Route Event to Appropriate Modeling Element Instance
Instance 1
Instance 2
Instance 3
...
Instance n
Schedule Next Events

Figure 8-8 Simulation Event Scheduling and Processing

8.2.3.3 Random Number Sampling

The core component of this service provides support for random number generation and transformation into several standard stochastic distributions. Actual transformed values are obtained through the Beta, Expntl, Normal, Triang, and Uniform methods, which generate beta, exponentially, normally, triangularly and uniformly distributed deviates, respectively. Other types of transformations can be supported through a user service as will be explained in Section 8.2.3.9.

Another layer above the core component is also available for supporting element attributes with an internal type set to a distribution. An attribute's internal representation can be set to the distribution type to allow users to modify the distribution specifications at run-time through the GUI (see 8.2.3.8.3) and then allow developers to sample random numbers based on the user's information. Several methods and properties, as listed in Table 8-2, are provided to allow developers to manually control these types of attributes.

8.2.3.4 Tracing

The tracing service provides support for a general purpose feedback management system. It is used by developers to insert tracing statements for tracking simulation progress or for reporting certain errors. The tracing service is also used internally by the system to report any errors encountered at run-time.

Developers can generate trace messages through a call to the Trace method of the Tracer object. The first argument is a string consisting of the message. The second argument specifies the message category, which the user can use as a filter to examine a specific category of messages. Two further subcategory levels can also be defined through third and fourth arguments. The convention is to use "Simulation" for simulation tracing messages, "Plan" for project plan type information, and "Integrity" for reporting integrity type errors. Internal run-time errors are generated with the category "Execution". An example of a run-time error is a division by zero. The "Execution" category is a special category because any generated trace messages will cause the simulation service to stop if it is currently executing a model.

For every trace call, the system will automatically track the current simulation run, simulation time, originating element instance, and element event. The last generated message can be accessed with the LastMessage property. The TraceEnabled property allows developers to toggle the status of the service. If its value is False then any generated tracing messages, with the exception of those based on the "Execution" category, will be ignored.

The GUI service provides a dialog window for intuitive navigation by users of generated trace messages (see Section 8.2.3.8.5).

8.2.3.5 Statistical collection and Analysis

This service provides the means for the automated collection and analysis of statistics during simulation. Analysis of observations collected with the Collect member method depends on whether the associated statistic was declared as intrinsic or non-intrinsic. For each data point that is collected, the simulation time at which it was collected is also tracked. A statistic declared as non-intrinsic will cause the analysis to ignore this time component and perform simple analysis based only on the collected data values. Otherwise, the calculations will be biased by the time value of the collected statistic. Intrinsic type statistics are typically used for variables that indicate a certain state of the model such as the current file length or the number of available resources while non-intrinsic statistics are used for performance type variables such as cycle time, travel time, and waiting time.

With intrinsic analysis, the observation time interval is extremely important. By default, the interval is defined by the minimum and maximum time at which the data was collected. This is valid most of time. But in some situations it might be necessary to manually set this interval to adjust the analysis. Consider, for example, a statistic that is tracking the utilization of a backhoe. The data points for this statistic are collected from the time that a certain process first obtains the backhoe until the time it is finally released. Based on this information, the default analysis will calculate the average utilization based on the first time the backhoe was requested until the last time that it was released. However, it might make more sense to analyze the backhoe utilization across the entire duration of the project and not only based on the activities in which it was involved. To do this, developers can choose to define the interval manually using the SetStart and SetFinish methods.

Statistical data can also be tracked in full or in summary. The tracking method is specified during the AddStatistic call and can later be examined with the FullTracking property. "Full tracking" means that the collected data is dumped to the database on each call and that the user can view a histogram or a time-graph representation of the data through the GUI (see section 8.2.3.8.4). The alternative is summary tracking, which means that detailed statistical data is not kept and only summary information is available. This type of tracking utilizes far less memory space and CPU time.

At any time during or after the simulation, several properties, as listed in Table 8-2, are available to allow developers to extract the current statistical analysis results. This includes the mean, standard deviation, minimum and maximum for a given run as well as global results across all runs.

8.2.3.6 Planning

The planning service allows for the generation of project planning information conforming to the standard previously discussed in Section 7.4.2. Activities are first declared for each modeling element as part of the planning behavior using the AddActivity member method. Once activities are declared, The AddResource method can be used to declare required resources, and the AddCostCenter method can be used to declare cost centers, which are used for tracking costs.

During the simulation execution, calls can be made to RecognizeProduction to recognize the production of a certain amount, to RecognizeRevenue to recognize that a certain compensation amount has been received, to RecongizeCost to recognize that cost for a certain cost center has been incurred, and OffsetResource to recognize the allocation or de-allocation of a certain resource.

For each planning related call, Simphony records the current simulation run and time and stores the data in the relational structure illustrated in Figure 7-5.

The GUI service provides several forms that display a summary of the generated planning information. These are presented in Section 8.2.3.8.6.

8.2.3.7 Database Access

The database access service provides support for accessing external databases. This service is based on Microsoft's Data Access Objects (DAO) library, which supports cursor or batch SQL-based commands to be executed on an ODBC compliant data source. The actual set of supported properties and methods is very extensive and as such, will not be discussed here. It is sufficient to say that developers can utilize this service to retrieve and manipulate information in external databases. For more information on ODBC and DAO, see Vaughn (1998).

8.2.3.8 Graphical User Interface (GUI)

8.2.3.8.1 General

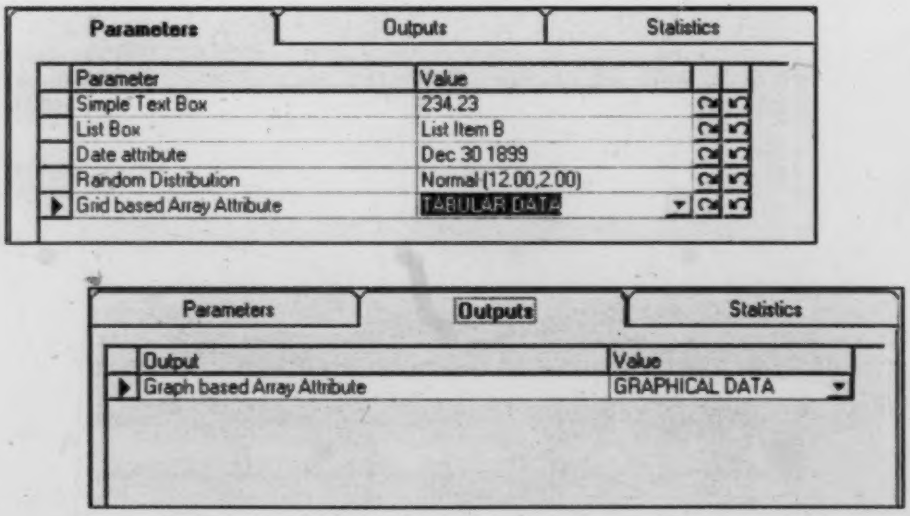
The GUI service provides all the required functionality to support a graphical user interface for controlling the modeling and simulation system. This includes a wide range of dialog boxes, forms, user interface controls, and line and bitmap primitives.

8.2.3.8.2 Graphical Drawing Primitives

Access to the graphical drawing primitives of the GUI service is available through the CDC statement. Examples of supported line drawing methods are MoveTo, LineTo, Circ, Rectangle, Ellipse, Arrow and ArrowHead. The format of the line output methods can be controlled with the ChangeLineStyle and the ChangeFillColor methods. The RenderPicture method is used to transfer a bitmap from the bitmap database onto the display surface. The bitmap database is managed by the developer and will be illustrated in Section 8.2.4.2. Text drawing is also supported through the TextOut, ChangeTextColor, and ChangeFont methods.

8.2.3.8.3 Attribute External Representation

The list of attributes for a given modeling element is viewed using the element attribute dialog box shown in Figure 8-9. Parameter type attributes are shown separately from output type attributes.



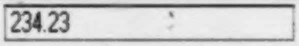
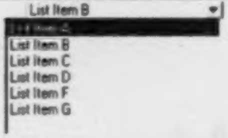
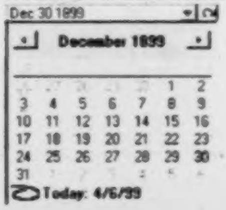
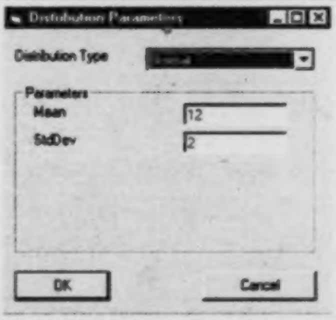
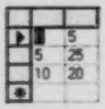
	Parameter	Value
	Simple Text Box	234.23
	List Box	List Item B
	Date attribute	Dec 30 1899
	Random Distribution	Normal (12.00,2.00)
▶	Grid based Array Attribute	TABULAR DATA

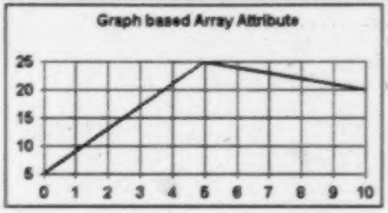
	Output	Value
▶	Graph based Array Attribute	GRAPHICAL DATA

Figure 8-9 Attribute Manipulation through the Element Attribute Dialog Box

The user interface control that is shown when the user attempts to view or modify the value of the attribute varies with the developer's choice of internal and external representation set during the initial call to the AddAttribute method. Table 8-3 shows a list of the possible options and a sample control.

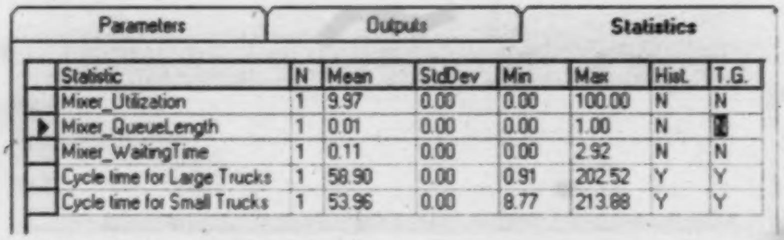
Table 8-3 GUI Controls Used for the External Representation of Element Attributes

Internal Rep.	External Rep.	GUI Control	Sample
Numeric, text	Default	Text Box	
Numeric, text	List box	List Box	 List Item B List Item B List Item C List Item D List Item F List Item G
DateTime	Default	Calendar	 Dec 30 1899 December 1899 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 Today: 4/6/99
Distribution	Default	Distribution Dialog Box	 Distribution Parameters Distribution Type Parameters Mean 12 StdDev 2 OK Cancel
Array	Table	Grid	

Array	Graph	Graph	

8.2.3.8.4 Statistical Analysis Representation

The statistical analysis results of collected data are also provided through the element attribute dialog box as shown in Figure 8-10. Included in the information shown are the number of runs for which data was collected, the global mean and standard deviation, and the minimum and maximum values observed. For statistics declared with full tracking, a histogram represenation (Figure 8-11) and a time graph (Figure 8-12) can also be viewed. Time graphs display the value of the statistic as a function of time.



	Statistic	N	Mean	StdDev	Min	Max	Hist.	T.G.
	Mixer_Utilization	1	9.97	0.00	0.00	100.00	N	N
▶	Mixer_QueueLength	1	0.01	0.00	0.00	1.00	N	[illegible]
	Mixer_WaitingTime	1	0.11	0.00	0.00	2.92	N	N
	Cycle time for Large Trucks	1	58.90	0.00	0.91	202.52	Y	Y
	Cycle time for Small Trucks	1	53.96	0.00	8.77	213.88	Y	Y

Figure 8-10 GUI Presentation of Summary Statistical Analysis Results

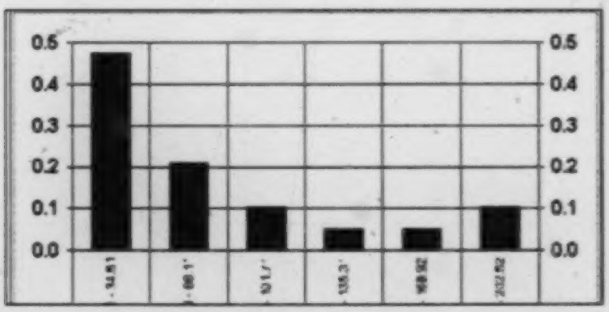


Figure 8-11 Sample Generated Histogram

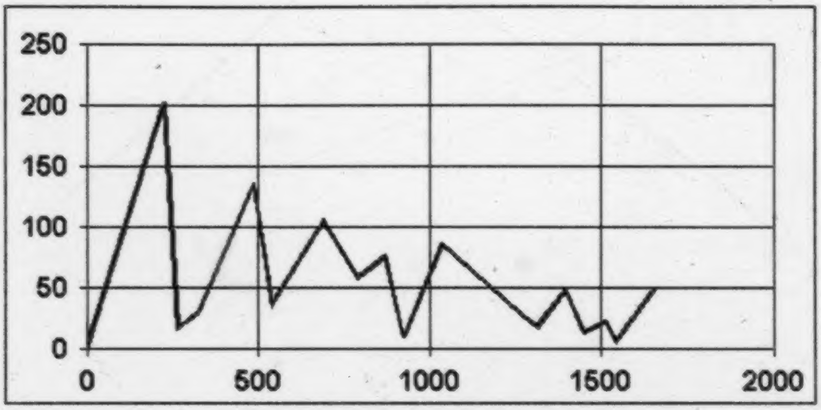
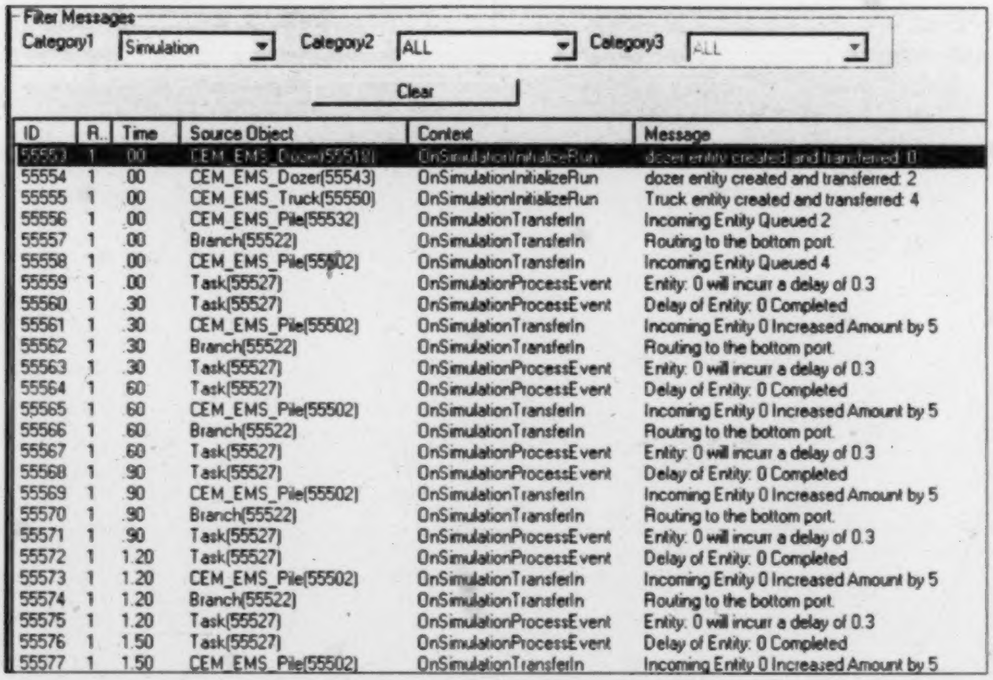


Figure 8-12 Sample Generated Time Graph

8.2.3.8.5 Trace Message Representation

The GUI service includes a special form that is used to view generated tracing messages. This form, shown in Figure 8-13, allows users to filter and sort the messages according to their categories.



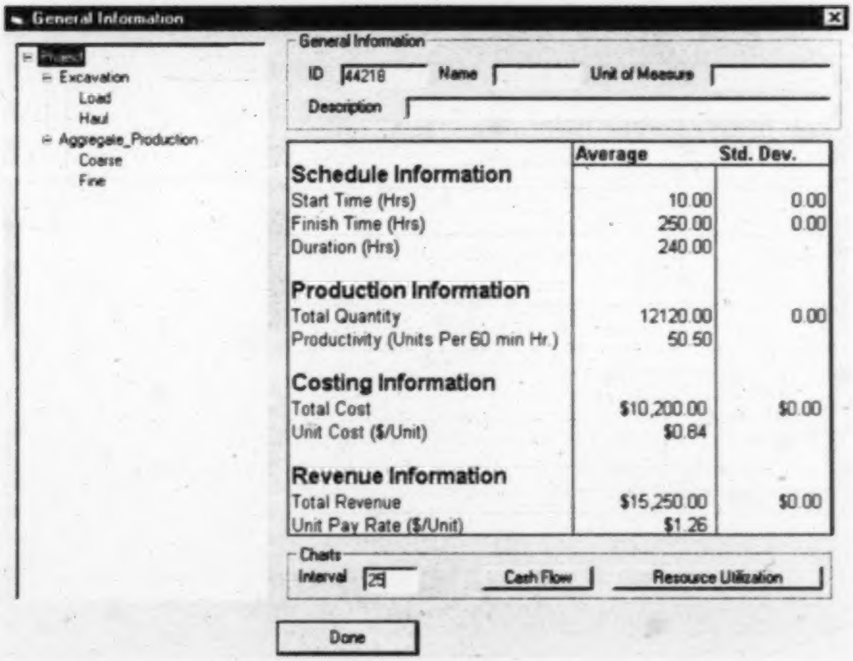
Category1: Simulation
Category2: ALL
Category3: ALL

ID	R..	Time	Source Object	Context	Message
55553	1	.00	CEM_EMS_Dozer(55518)	OnSimulationInitializeRun	dozer entity created and transferred: 0
55554	1	.00	CEM_EMS_Dozer(55543)	OnSimulationInitializeRun	dozer entity created and transferred: 2
55555	1	.00	CEM_EMS_Truck(55550)	OnSimulationInitializeRun	Truck entity created and transferred: 4
55556	1	.00	CEM_EMS_Pile(55532)	OnSimulationTransferIn	Incoming Entity Queued 2
55557	1	.00	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port.
55558	1	.00	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity Queued 4
55559	1	.00	Task(55527)	OnSimulationProcessEvent	Entity: 0 will incurr a delay of 0.3
55560	1	.30	Task(55527)	OnSimulationProcessEvent	Delay of Entity: 0 Completed
55561	1	.30	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5
55562	1	.30	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port.
55563	1	.30	Task(55527)	OnSimulationProcessEvent	Entity: 0 will incurr a delay of 0.3
55564	1	60	Task(55527)	OnSimulationProcessEvent	Delay of Entity: 0 Completed
55565	1	.60	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5
55566	1	.60	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port.
55567	1	.60	Task(55527)	OnSimulationProcessEvent	Entity: 0 will incurr a delay of 0.3
55568	1	.90	Task(55527)	OnSimulationProcessEvent	Delay of Entity: 0 Completed
55569	1	.90	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5
55570	1	.90	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port.
55571	1	.90	Task(55527)	OnSimulationProcessEvent	Entity: 0 will incurr a delay of 0.3
55572	1	1.20	Task(55527)	OnSimulationProcessEvent	Delay of Entity: 0 Completed
55573	1	1.20	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5
55574	1	1.20	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port.
55575	1	1.20	Task(55527)	OnSimulationProcessEvent	Entity: 0 will incurr a delay of 0.3
55576	1	1.50	Task(55527)	OnSimulationProcessEvent	Delay of Entity: 0 Completed
55577	1	1.50	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5

Figure 8-13 Trace Navigation Form

8.2.3.8.6 Project Plan Representation

Earlier, it was mentioned that project-planning information is generated primarily for use by external systems. However, it is believed that users can also benefit from the ability to quickly view a summary of the generated information in a simple format prior to the final analysis by external systems. As a result, the GUI service includes a number of project planning display forms. The main form is shown in Figure 8-14. The left side displays a tree that contains a hierarchical view of the project activities. At the highest level is the project itself. Selecting a given activity from this tree will display its related information on the sheet shown on the right.


General Information
Project
Excavation
Load
Haul
Aggregate_Production
Coarse
Fine
General Information
Cash Flow
Resource Utilization
Done

ID 44218
Name
Unit of Measure
Description

	Average	Std. Dev.
Schedule Information		
Start Time (Hrs)	10.00	0.00
Finish Time (Hrs)	250.00	0.00
Duration (Hrs)	240.00	
Production Information		
Total Quantity	12120.00	0.00
Productivity (Units Per 60 min Hr.)	50.50	
Costing Information		
Total Cost	$10,200.00	$0.00
Unit Cost ($/Unit)	$0.84	
Revenue Information		
Total Revenue	$15,250.00	$0.00
Unit Pay Rate ($/Unit)	$1.26	

Charts
Interval 25

Figure 8-14 Project Plan Summary Form

The schedule information includes the start and finish time of the activities in terms of elapsed hours. The activity duration is simply the difference between the finish and start times. The production information section shows the total quantity recognized for the selected activity and the resultant productivity based on the total quantity and the activity duration. The costing information section shows the total cost incurred and the unit cost based on the recognized total quantity. Similarly, the revenue information section shows total revenue generated and the unit pay rate based on total revenue and total quantity recognized.

Below the summary sheet, two buttons are available. The first, "Cash Flow", presents a secondary form with a chart that shows the total revenue, total cost and net income as a function of the simulation time. The second, "Resource Utilization", activates another form that displays the various resources utilized as a function of the simulation time.

8.2.3.9 User Services

The presented services provide commonly used routines for supporting the developer in creating a new special purpose simulation template. However, Simphony was designed in such a manner that extra user services can be added to the existing service pool. Advanced developers or third party specialists can implement new specialized services for integration with the Simphony development environment. Examples of potentially useful services include a comprehensive random sampling library, a CAD access library, an artificial neural network training and recall library, and an optimization library.

8.2.4 The Simphony Designer Program

8.2.4.1 Introduction

The Simphony Designer is an integrated development environment that allows for the definition of modeling elements. It allows developers to define the element behaviors using a language based on Visual Basic for Applications (VBA). VBA was developed by Microsoft Corporation for use as macro-based extension language for its series of office productivity tools including Access, Excel and Word. The decision to use VBA was driven by the fact that numerous major developers now use it as the standard macro support language within their applications. This includes such products as AutoCAD and Primavera. The main form for the program is shown in Figure 8-15.

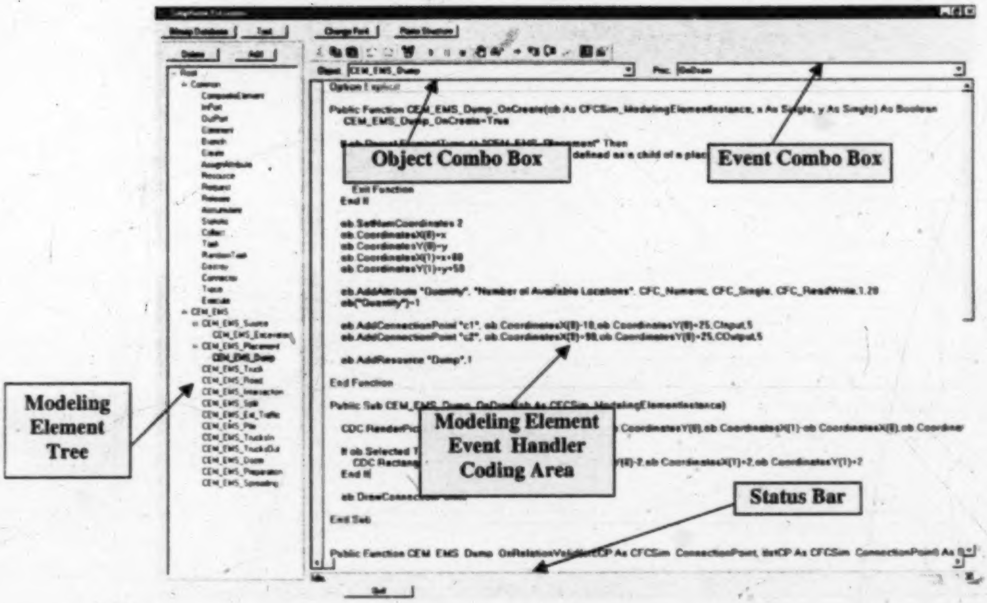

Object Combo Box
Event Combo Box
Modeling Element Tree
Modeling Element Event Handler Coding Area
Status Bar

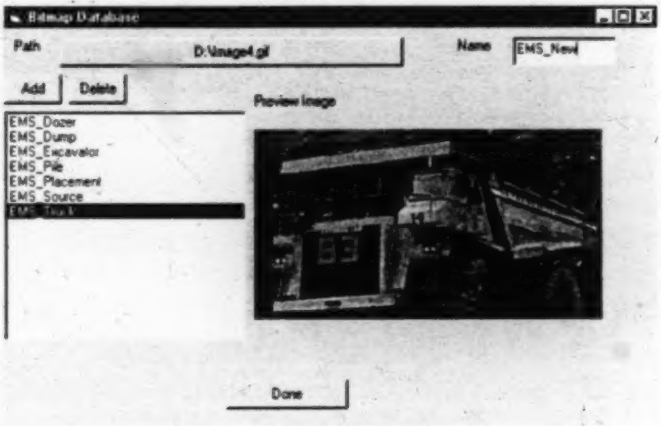
Figure 8-15 Main Form of Simphony Designer

The modeling element tree displays a list of all currently defined modeling elements in the modeling element library. A tree view is used because the structure of the library is hierarchical. Clicking on a given modeling element from the tree displays its event handlers in the "Event handler Coding Area" and allows developers to change their definition or insert new event handlers.

Developers can save their work at any time during the implementation process. Before saving, Simphony Designer will check the code and ensure that there are no syntax errors. If any errors exist, they will be explained in the status bar and a red line will indicate the position of the error.

8.2.4.2 The Bitmap Database

Simphony Designer includes a bitmap database that stores all the pictures that are available for use by the modeling elements for customizing their graphical representation through the RenderPicture method. The form used to manipulate this database is shown in Figure 8-16.



Path D:\image4.gif

Name EMS_New

EMS_Dozer
EMS_Dump
EMS_Excavator
EMS_Pile
EMS_Placement
EMS_Source
EMS_Truck

Figure 8-16 Bitmap Database Manipulation Form

8.2.5 Template Design Guidelines

The primary objective of Simphony developers is the construction of a tool that can be used by construction practitioners to automate, formalize and verify their decision making process. The first step in the development of a special purpose simulation tool consists of a careful and detailed analysis of the target construction process. It is very important for developers to understand in detail the engineering and management fundamentals, resources, interactions and transformations that occur within that process and to keep in mind the overall objective of the tool. This can be accomplished through a series of interviews with engineers and experts who are well experienced highly the target process. After a thorough analysis and documentation of the process and the engineer's requirements, the design of the tool can begin.

The best design is the one that can take the most advantage of Simphony's ability to allow for the construction of re-usable simulation models. Reusability is accomplished through a combination of parameterized models and flexible modeling elements. With parameterized models, element parameters can be modified by users at run-time to simulate slightly different scenarios and generate different outcomes. Flexible modeling elements take advantage of modularity concepts to allow certain elements to be reused in several contexts. The end result, if designed correctly, is a tool that allows for the modeling of a large proportion of all the possible scenarios within the target domain.

After the completion of the initial investigation, the detailed SPS template design can begin. This involves the identification of the required modeling elements followed by the definition of the behaviors of each modeling element. The number of modeling elements required depend greatly on the nature of the modeled construction process and the amount of reusability required. In most cases, a standard list of modeling elements will first be defined and used. This list could gradually grow to allow for the modeling of more situations depending on the user's requirements. Table 8-4 provides a step-by-step process that developers can use as a guideline for obtaining the initial set of modeling elements.

Table 8-4 Simphony Template Design Guidelines

Step	Action	Comment	Objective
1	Provide a general description of the target construction process	What are we trying to analyze? What are the resources involved? (Labor, Equipment,..) What is the overall measure of performance What is a typical scenario? What is the general nature of variations from one scenario to the next?	The end result of this analysis will be the Primary Modeling Element that represents the construction operation as a whole
2	Define the	Based on the identified scenario	The identified basic elements

	elements of the construction process	variations, identify the basic elements involved in the target process.	will translate into the Child Elements of the Primary Modeling Element.
3	Define the function and behavior of each construction process element	Based on the identified elements, define the preliminary list of attributes and operations that each element will represent.	This preliminary process definition for each element will be the basis for the implementation of the behaviors of the modeling element.

8.2.6 Template Implementation Guidelines

Table 8-5 presents a step-by-step guideline for the development of a modeling element after the completion of its design. The illustrated process encourages the incremental development and testing of the modeling element behaviors.

Table 8-5 Guidelines for the Incremental Development of a Modeling Element

Description		Comments	Events
1	Define Initial Attributes	Use the AddAttribute method to add the required read-write attributes of the modeling element.	OnCreate
2	Define Relations	Use the AddConnectionPoint method to add the required connection points of the modeling element.	OnCreate
3	Simulation	Define Simulation Behavior	
4 5	Declare any required internal attributes	Use the AddAttribute method to add the required hidden attributes of the modeling element.	OnCreate
6	Declare any required resources	Use the AddResource method to add any required simulation resources.	OnCreate
7	Declare any required files	Use the AddFile method to add any required simulation files.	OnCreate
8	Declare any required simulation events	Use the AddEvent method to add any required simulation events.	OnSimulationInitialize
9	Set the resource quantities	Use the NumResources property of the resource object to set the number of resources based on the corresponding attribute values if needed.	OnSimulationInitialize
10	Declare any starting entities and schedule their starting events	Use the AddEntity method to create the initial model entities and schedule their starting events with the ScheduleEvent method.	OnSimulationInitializeRun

11 Process the simulation events	Write the appropriate code to process the simulation events. Use a "Select-Case" statement to filter the events.	OnSimulationProcessEvent
12 Intercept incoming entities	Intercept incoming entities in order to implement manual routing or to do some pre-processing if needed.	OnSimulationTransferIn
13 Define integrity rules		
13.1 Define parameter integrity	If the default parameter integrity checking is not sufficient, implement your own. Parameter integrity checking is performed after the user modifies any of the attribute values in the element property form.	OnValidateParameters
13.2 Define genealogy-based integrity	In your implementation of the OnCreate event handler, return False if the element violates hierarchical integrity. For example, you might choose to allow the creation of truck elements only as children of the Move_Earth element.	OnCreate
13.3 Define relationship integrity	Implement any relationship-based integrity. For example, you might want to limit the number of relations from a given output connection point or constraint the types of the elements which could be connected.	OnRelationValid
13.4 Define model level integrity	Implement any model level checks that cannot be done with the previous three types of integrity checks. Model level integrity is performed when the engineer requests that the simulation be initiated. An example of a check you might want to do here is ensuring that that an earth-moving element contains at least one source child element.	OnCheckIntegrity
14 Define statistics		
14.1 Declare any required custom statistics	Use the AddStatistic method to declare any required statistics such as truck cycle times. Note that resources and files come with several default statistics.	OnCreate
14.2 Collect statistics	Use the Collect method of the statistic object to record observations. This should typically be done in the OnSimulationProcessEvent event handler.	OnSimulationProcessEvent OnSimulationTransferIn
15 Declare any useful output attributes	Use the AddAttribute method to declare any read-only attribute that could be beneficial to the engineer.	OnCreate

16 Customize graphical Representation	Add any bitmaps you might need to the bitmap database. Use these bitmaps in order to improve the graphical appeal of the modeling elements. Alternatively, you can use any of the available line and text output methods of the graphics service. If you need more than a single reference point, call the SetNumCoordinates method.	OnCreate OnDraw OnDrawRelation OnDragDraw
17 Customize graphical Manipulation	If the default graphical manipulation of the element is not satisfactory, implement your own. You might want to do this if you need to support rotation or highly non-rectangular element representations.	OnGetBoundingBox OnHitTest OnMove
18 Define planning		
18.1 Declare activities, activity resources, and activity cost centers.	Use the AddActivity method to declare any activities that should be generated through the current modeling element.	OnCreate
18.2 Recognize cost	RecognizeCost	OnSimulationProcessEvent
18.3 Recognize revenue	RecognizeRevenue	OnSimulationProcessEvent
18.4 Track resource levels	OffsetResourceLevel	OnSimulationProcessEvent

8.2.7 Sample Tool Development Session

8.2.7.1 Overview

An example will now be provided to illustrate how Simphony is used to create a simple modeling element. The new element will allow a plant owner to model a simple concrete batch plant operation. Incoming trucks arrive at the plant site and wait for their turn. When the mixer is available, the next truck in the queue loads a certain quantity of concrete and then proceeds to its destination.

The Simphony Designer program is first used to create a new element called "Batch_Plant". Next, the simulation behavior of the new element is defined to model the described scenario.

8.2.7.2 Step 1 : Building a Working Model

Simulation model entities representing the trucks will be needed. So will a resource to represent the mixer location. The discrete simulation events will be: TruckArrive, RequestMixer, and ReleaseMixer. The mixer resource is declared by implementing a handler for the OnCreate event as following:

```
Public Function Batch_Plant_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    Batch_Plant_OnCreate = True
    ob.OnCreate x,y,True

    ob.AddResource "Mixer",1
End Function
```

The first line inside the handler sets the return value to True to instruct Simphony to create the element. If for some reason the element should not be created, the return value can be set to False. The second line calls the default implementation of the event handler. The third line declares the mixer resource with an initial quantity of one. Next, the needed events are declared as part of the developer's implementation of the OnSimulationInitialize event handler:

```
Public Sub Batch_Plant_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "TruckArrive", True
    ob.AddEvent "RequestMixer"
    ob.AddEvent "ReleaseMixer"
End Sub
```

A starting truck will need to be initialized at the beginning of the simulation. At each truck's arrival, the arrival of the following truck is scheduled. In order to initialize the first truck in the model, the OnSimulationInitializeRun event , which is triggered at the beginning of each simulation run, is implemented as following:

```
Public Sub Batch_Plant_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance, RunNum
As Integer)
    Dim truck As CFCSim_Entity
    Set truck = ob.AddEntity

    ob.ScheduleEvent truck, "TruckArrive",0
End Sub
```

The above code first declares a variable of type CFCSim_Entity. The second line calls the AddEntity method to obtain a reference to a new entity, which is then assigned to the declared variable. On the third line, the ScheduleEvent method is used to schedule the first event for the starting entity. The ScheduleEvent method expects the entity as a first parameter. The second parameter is the event name to be scheduled and the third parameter is the duration from the current simulation time at which the event should occur.

The main simulation processing code is defined in OnSimulationProcessEvent event handler. For the batch_Plant, this is done as following:

```
Public Sub Batch_Plant_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent
As String, Entity As CFCSim_Entity)
    Dim NewTruck As CFCSim_Entity

    Select Case MyEvent
    Case "TruckArrive"
        'schedule the arrival of the next truck
        Set NewTruck = ob.AddEntity
        ob.ScheduleEvent NewTruck,"TruckArrive",20.0

        'schedule the next event for the current truck
        ob.ScheduleEvent entity,"RequestMixer",0.0

    Case "RequestMixer"
        If ob.RequestResource("Mixer",entity) Then

            ob.ScheduleEvent entity,"ReleaseMixer",15.0
        End If
    Case "ReleaseMixer"
        ob.ReleaseResource "Mixer",entity

        ob.DeleteEntity entity
    End Select
End Sub
```

The parameters of the event provide the event name that must be processed as well as a reference to whichever entity originally scheduled the event. For the "TruckArrive" event, the above code first creates another truck entity represent the next truck that will arrive then schedules its arrival time at 20 minutes. This means that currently, the assumption is made that the inter-arrival time of the trucks is 20 minutes. Next the code schedules the "RequestMixer" event for the current truck. When the current event is the "RequestMixer", the above code first calls the RequestResource method to obtain the mixer resource. If the mixer is available, then the method call will return a True value. Otherwise a False value will be returned and both the current event and the requesting entity are saved and automatically added to the resource queue. When the mixer becomes available, the saved event will be automatically rescheduled for the saved entity. When the mixer is obtained, the "ReleaseMixer" event is scheduled to occur in 15 minutes. The assumption here is that loading time will be 15 minutes. When the "ReleaseMixer" event is to be processed, the above code first calls the ReleaseResource method to release the mixer and then destroys the current truck entity as it is no longer needed.

At this point a working tool that models the described situation has been defined. It can be tested using the Simphony Editor program and statistics can be examined for the mixer resource.

8.2.7.3 Step 2 – Adding Randomness

The second step will demonstrate how randomness can be incorporated into the simulation code through the random number sampling service. The assumption is now that, instead of trucks arriving every 20 minutes, the inter-arrival time will be

?

exponentially distributed with a mean of 20 minutes. Similarly, the loading time will be normally distributed with a mean of 15 minutes and a standard deviation of 2 minutes. To do this, the implementation of the OnSimulationProcessEvent is modified as follows:

<u>Before:</u>

```
...
ob.ScheduleEvent NewTruck,"TruckArrive",20.0
...
ob.ScheduleEvent entity,"ReleaseMixer",15.0
...
```

<u>After:</u>

```
...
ob.ScheduleEvent NewTruck,"TruckArrive",Sampler.expntl(20.0)
...
ob.ScheduleEvent entity,"ReleaseMixer",Sampler.normal(15.0,2)
...
```

Instead of supplying direct deterministic values, the above code calls the sampling service to obtain a random number.

8.2.7.4 Step 3 – Incorporation of Different Truck Sizes

The next step is to demonstrate how trucks of different sizes can be incorporated into the model. An assumption is made that, on average, half the incoming trucks will be large and half will be small. For large trucks, the loading time is Normal(25,2); for small trucks it is Normal(15,3). To do this, an entity attribute will be used to determine the size of the truck. For the initial entity created at the beginning of the simulation run, it is assumed to be large:

```
Public Sub Batch_Plant_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance, RunNum As Integer)
    Dim truck As CFCSim_Entity
    Set truck = ob.AddEntity

    truck("Size")="Large"
    ob.ScheduleEvent truck, "TruckArrive",0
End Sub
```

The main change highlighted above initializes a new entity attribute called "Size" and assigns it the value "Large". Next, a similar change is made to the OnSimulationProcessEvent event handler as follows:

```
    Dim x as single
    ...
    Select Case MyEvent
    Case "TruckArrive"
        ' schedule the arrival of the next truck
        Set NewTruck = ob.AddEntity
        x = Sampler.uniform(0,1)
        If x < 0.5 Then
            NewTruck("Size")="Small"
        Else
            NewTruck("Size")="Large"
        End If

        ob.ScheduleEvent NewTruck,"TruckArrive",Sampler.expntl(20.0)
    ...
```

The above code first declares a single precision floating point variable (Single). When the "TruckArrive" event is handled, the above code samples a random uniform number between 0 and 1. If this number is less than 0.5, it sets the truck size to "Small" otherwise, it sets it to "Large". This has the effect of generating an equal number of small and large trucks.

The other change required will be to the processing code for the "RequestMixer" simulation event. Changes are as follows:

```
    Case "RequestMixer"
        If ob.RequestResource("Mixer",entity) Then

            If entity("Size")="Large" Then
                ob.ScheduleEvent entity,"ReleaseMixer",Sampler.normal(25,2)
            Else
                ob.ScheduleEvent entity,"ReleaseMixer",Sampler.normal(15,3)
            End If

        End If
```

The above code first checks the value of the entity's "Size" attribute and, depending on its value, the appropriate parameters of the distribution are used.

8.2.7.5 Step 4 – Using Parameter Attributes

Thus far, the element definition includes numerous assumptions about the batch plant operation. This means that if the owner of the batch-plant company would like to experiment with different truck inter-arrival times or loading duration, they must either posses enough information to modify the code, which is highly unlikely, or contact the developer to make the necessary changes. This is obviously unpractical; what is needed instead is a parameterized modeling element. This means that attributes for the modeling element must be defined, which the plant owner can change in the Simphony Editor. In order to provide the plant owner with control over the truck loading duration, two attributes are added to the modeling element as follows:

```
Public Function Batch_Plant_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    Batch_Plant_OnCreate=True
    ob.OnCreate x,y,True

    ob.AddAttribute "LoadLarge","Loading Duration for Large Trucks",CFC_Distribution,
CFC_Single, CFC_ReadWrite
    ob.AddAttribute "LoadSmall","Loading Duration for Small Trucks",CFC_Distribution,
CFC_Single, CFC_ReadWrite

    ob.AddResource "Mixer",1
End Function
```

The changes highlighted above use the AddAttribute method to declare two attributes of type "CFC_Distribution". This means that users will be presented with a special dialog box where they can choose the distribution type and set its parameter values. Once the attributes are declared, their values will need to be used as part of the simulation processing. To do this, the OnSimulationProcessEvent event handler is changed as follows:

```
If entity("Size")="Large" Then
    ob.ScheduleEvent entity,"ReleaseMixer",ob("LoadLarge")
Else
    ob.ScheduleEvent entity,"ReleaseMixer",ob("LoadSmall")
End If
```

Instead of specifying a value directly, the above code simply references the declared attribute, which returns a random value based on the user's specifications. The attribute values can be manipulated through modeling element attribute form shown in Figure 8-9.

8.2.7.6 Step 6 – Adding Custom Statistics

The statistics available by default are those that are associated with the resources. If any extra custom statistics are needed, they must first be declared and the observation values collected during the simulation. Assuming that the owner is interested in the total service time of the truck, a new statistic is declared by adding the following line to the end of the OnCreate event handler:

```
ob.AddStatistic "CycleTime"," Truck Cycle Time",False, True
```

The shown statement declares a new, non-intrinsic statistic called "CycleTime" with full tracking. Now that the statistic is declared, the observations for it can be collected. To do this, the truck entity must first be tagged before it requests a mixer with a timestamp. This will allow for the determination of its cycle time upon completion of the loading operation. This can be done by making the following highlighted changes to the OnSimulationProcessEvent event handler:

```
Public Sub Batch_Plant_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent
As String, Entity As CFCSim_Entity)
    Dim NewTruck As CFCSim_Entity
    Dim x As Single
    Select Case MyEvent
    Case "TruckArrive"
        ' schedule the arrival of the next truck
        Set NewTruck = ob.AddEntity
        x = Sampler.uniform(0,1)
        If x < 0.5 Then
            NewTruck("Size")="Small"
        Else
            NewTruck("Size")="Large"
        End If

        ob.ScheduleEvent NewTruck,"TruckArrive",Sampler.expntl(20.0)

        entity("StartTime")=SimTime

        ' schedule the next event for the current truck
        ob.ScheduleEvent entity,"RequestMixer",0

    Case "RequestMixer"
        If ob.RequestResource("Mixer",entity) Then

            If entity("Size")="Large" Then
                ob.ScheduleEvent entity,"ReleaseMixer",ob("LoadLarge")
            Else
                ob.ScheduleEvent entity,"ReleaseMixer",ob("LoadSmall")
            End If

        End If
    Case "ReleaseMixer"
        ob.ReleaseResource "Mixer",entity

        ob.stat("CycleTime").Collect SimTime - entity("StartTime")

        ob.DeleteEntity entity
    End Select
End Sub
```

8.2.8 Development of the Common Template

8.2.8.1 Overview

In order to test the expressiveness and flexibility of Simphony, a set of modeling elements were developed to allow for the construction of simulation models based on the traditional general purpose simulation modeling approach. Table 8-6 lists the developed elements and their function. The actual development code is provided in Appendix 1.

Table 8-6 List of Developed Modeling Elements Used to Support GPS-Based Modeling

Modeling Element	Purpose	Parameters	Statistics
CreateEnt	Used to create entities.	Number of Entities Time of First Create Time Between Creates	
Destroy	Destroys incoming entities		
SetAttribute	Initializes new entity attributes or updates the values of an existing attribute.	Attribute 1 Name Attribute 1 Value ... Attribute 5 Name Attribute 5 Value	
Branch	Implements conditional and probabilistic branching.	Branching Probability	
Consolidate	Consolidates incoming entities into one entity.	Number to Consolidate Number to Generate	
Task	Delays entities by a deterministic value.	Delay Duration	
RandomTask	Delays entities by a random value.	Delay Duration	
Resource	Declares a resource for use with Capture and Release elements.	Resource Description Number of Resources	Utilization
Capture	Delays incoming entities until a given resource is available.	Resource Name Number of Resources Capture Priority	File Length Waiting Time
Release	Releases a previously captured resource.	Resource Name Number of Resources	
Statistic	Declares a user statistic to be tracked using the CollectStat element.	Statistic's name Intrinsic Status Full Tracking	User Statistic
CollectStat	Collects observed values. Used in conjunction with the Statistic element.	Statistic Value to Collect	
Connector	A dummy element used to organize relationship representation.		
Trace	Used to generate trace messages.	Expression Category	
Execute	Used to execute user code (user inserts).	Expression	
InPort	Used to route entities arriving at an input connection point of the parent element.		
OutPort	Used to route entities to an output connection point of a		

	parent.		
CompositeElement	A simple element designed to act as a container for other elements.		

The listed elements make up the bulk of the functionality that is often provided with a general-purpose simulation system. Users can utilize these elements to extend the existing set of SPS templates and build accurate and representative models. Three special elements, the InPort, OutPort and CompositeElement, have been added to allow for the creation of sub-models. This is based on Simphony's support for hierarchy and modularity concepts.

8.2.8.2 The Common Template as a Development Support Tool

Through the common template, users can in fact create elements of their own and place them in the User Model Library for later use. This will be illustrated in Section 8.3.2. Developers can also use this library to automate the definition of the simulation behavior of their elements. The previous approach was to implement the behavior explicitly by handling simulation-related events and implementing the required discrete event simulation code. A quicker alternative is to delegate the simulation behavior to a set of child elements, which are based on the common template. Incoming entities at the parent element would simply be routed for processing by the child elements. Communication between parent and child elements is supported with the InPort and the OutPort nodes.

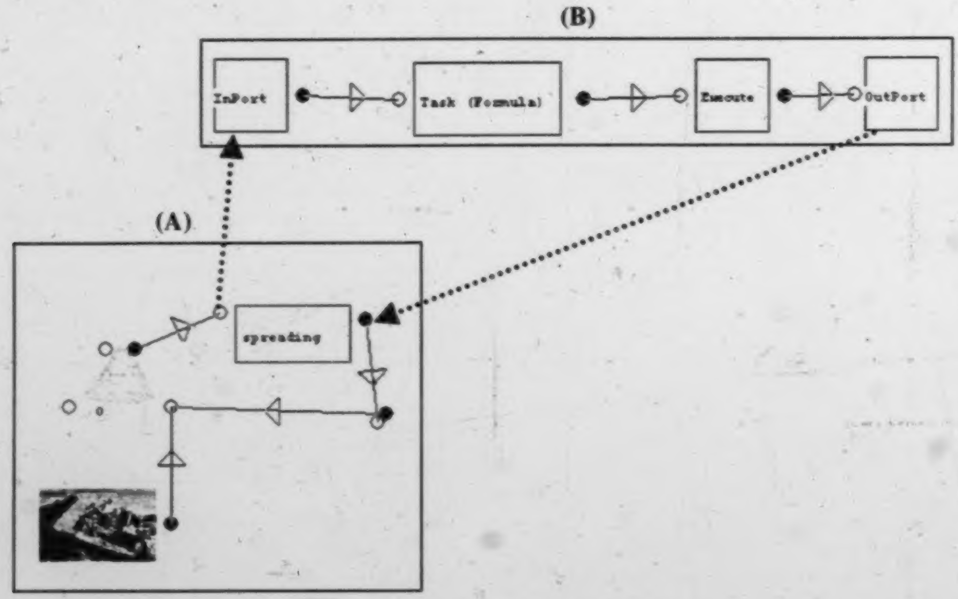


Figure 8-17 Utilization of the Common Template to Define Simulation Behavior

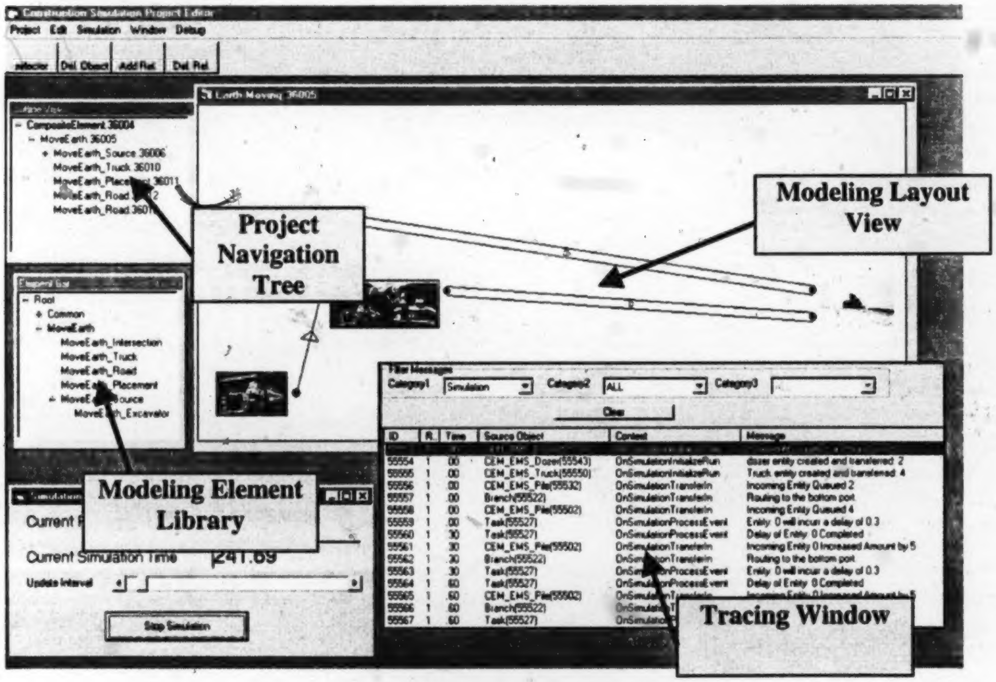
Figure 8-17A contains a model of a spreading operation. A dozer travels to a spreading pile, removes a certain quantity and then performs the spreading operation. The dozer and the spreading pile elements are atomic in the sense that their internal structure is defined using code and not through other lower level elements. The spreading modeling element, in contrast, defines its simulation behavior through its children. Entities arrive at its input connection point and are routed to the sub-model shown in Figure 8-17B. The sub-model is built using a set of common elements. The final element, OutPort, transfers the entity back to the higher level model.

The actual structure of the sub-model can be prototyped using the modeling environment to accomplish the required behavior and then later translated into the equivalent script. The end result is a modeling element that creates a set of child elements as soon as it is created.

8.3 Construction Simulation using the Simphony Editor

8.3.1 Introduction

Simphony Editor is a computer system that allows users to create and execute simulation models based on the elements available in the modeling element library. It acts as the controller of Simphony's GUI service. The main form, shown in Figure 8-18, is based on a multiple document interface (MDI) standard. This allows multiple windows to be open at the same time in order to represent different views of the model.


Construction Simulation Project Editor
Project Edit Simulation Window Debug
Del Object
Add Rel
Del Rel
Earth Moving 36005
- CompositeElement 36004
- MoveEarth 36005
+ MoveEarth_Source 36006
MoveEarth_Truck 36010
MoveEarth_Placement 36011
MoveEarth_Road 36012
MoveEarth_Road 3601
Project Navigation Tree
Modeling Layout View
Element Bar
- Root
+ Common
- MoveEarth
MoveEarth_Intersection
MoveEarth_Truck
MoveEarth_Road
MoveEarth_Placement
+ MoveEarth_Source
MoveEarth_Excavator
Modeling Element Library
Current
Current Simulation Time
241.69
Update Interval
Stop Simulation
Filter Messages
Category1
Simulation
Category2
ALL
Category3
Clear
Tracing Window

ID	R.	Time	Source Object	Context	Message
55554	1	.00	CEM_EMS_Dozer(55543)	OnSimulationInitializeRun	dozer entity created and transferred 2
55555	1	.00	CEM_EMS_Truck(55550)	OnSimulationInitializeRun	Truck entity created and transferred 4
55556	1	.00	CEM_EMS_Pile(55532)	OnSimulationTransferIn	Incoming Entity Queued 2
55557	1	.00	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port
55558	1	.00	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity Queued 4
55559	1	.00	Task(55527)	OnSimulationProcessEvent	Entity 0 will incur a delay of 0.3
55560	1	.30	Task(55527)	OnSimulationProcessEvent	Delay of Entity 0 Completed
55561	1	.30	CEM_EMS_Pile(55502)	OnSimulationTransferIn	Incoming Entity 0 Increased Amount by 5
55562	1	.30	Branch(55522)	OnSimulationTransferIn	Routing to the bottom port
55563	1	.30	Task(55527)	OnSimulationProcessEvent	Entity 0 will incur a delay of 0.3
55564	1	.60	Task(55527)	OnSimulationProcessEvent	Delay of Entity 0 Completed
55565	1	.60	CEM_EMS_Pile(55502)	OnSimulationT	
55566	1	.60	Branch(55522)	OnSimulationT	
55567	1	.60	Task(55527)	OnSimulation	

Figure 8-18 Simphony Editor Main Form

The modeling element library displays all available elements that can be used to construct new models. The model layout window displays the current set of defined elements at a certain level in the project hierarchy. Displayed elements can be selected, deleted, edited and linked. Viewing the contents of one of the displayed elements will open another layout window. The project navigation tree displays the structure of the simulation project. Double-clicking on an item on the tree will bring up a model layout window for the element represented by the tree entry. The trace window is used to display and filter generated messages. The tool bar displayed at the top of the main form is used to delete selected elements, add and delete relationships, and change the zoom setting for the active model layout window. Double-clicking on a given element instance displays the modeling element attribute dialog box previously mentioned in Section 8.2.3.8.

8.3.2 User Model Library

The user model library is a component of the Simphony Editor; it allows users to store commonly used simulation models. It is analogous to a style sheet or a template in a word processing application. When a user adds an element to the user library, the element information, including its attribute values and its child elements, are stored. The stored element can later be retrieved to any simulation project. This will retrieve the child elements and also restore all the attribute values.

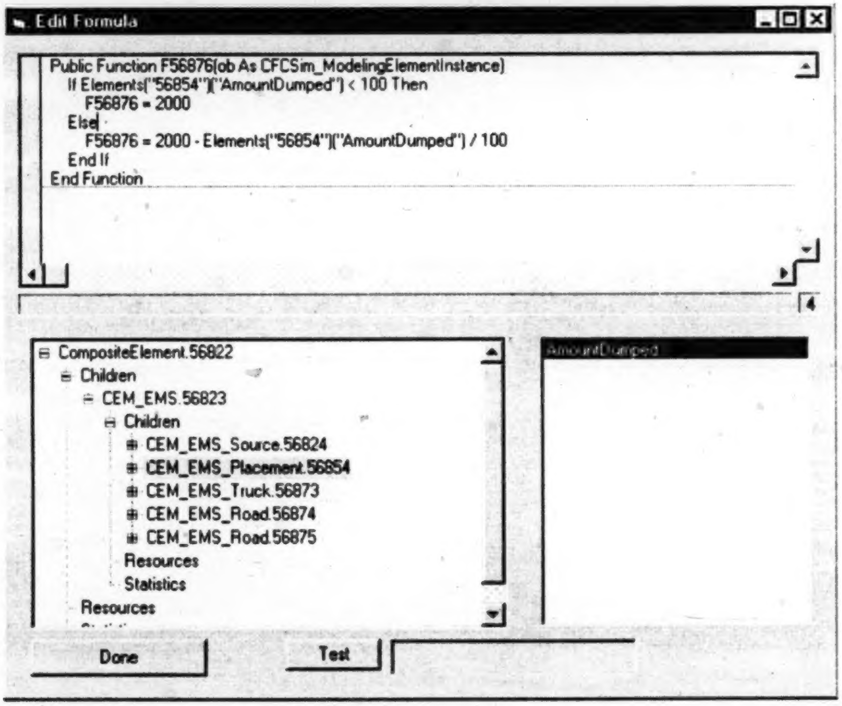
Users can in fact use this library as an element development tool. This is done by users first creating sub-models based on the common template to represent a given process and then store the defined model in the library. At any later time, this "user element" can be added to any simulation project.

8.3.3 Support for User Inserts

Figure 8-9 previously illustrated how users can manipulate parameter attribute values through the attribute dialog box. Users can instead choose to link a given parameter attribute to an expression. This expression is evaluated every time the attribute is accessed. The programming language used to build these expressions is based on VBA, which is the same language developers must use to define element behaviors.

Users have access to the entire Simphony object model. This means that they can access or even manipulate any component of the model including the current entity, attribute values of the current element or other elements within the simulation project.

Expressions are defined using the Expression Editor shown in Figure 8-19. The Expression Editor includes an area where the expression can be defined as well as a hierarchical view of the current project structure for supporting the construction of statements. The shown code is used to implement a road segment element with a dynamic length.



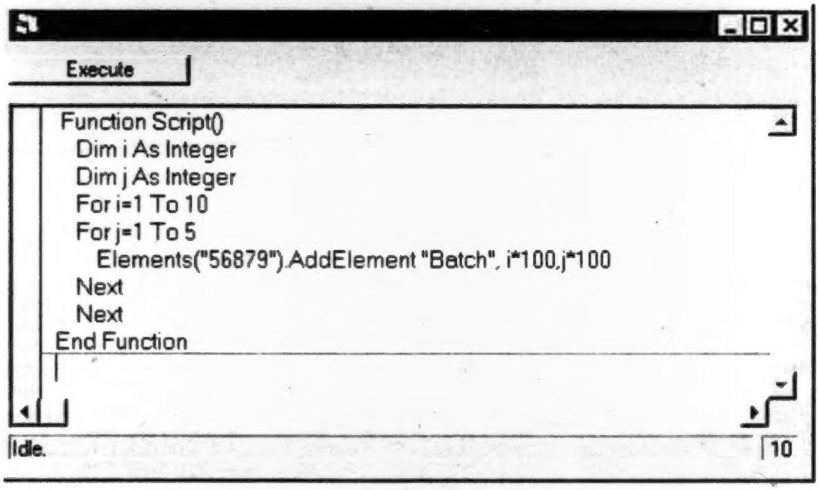
```
Public Function F56876(ob As CFCSim_ModelingElementInstance)
  If Elements("56854")("AmountDumped") < 100 Then
    F56876 = 2000
  Else
    F56876 = 2000 - Elements("56854")("AmountDumped") / 100
  End If
End Function
```

Edit Formula
CompositeElement.56822
Children
CEM_EMS.56823
Children
CEM_EMS_Source.56824
CEM_EMS_Placement.56854
CEM_EMS_Truck.56873
CEM_EMS_Road.56874
CEM_EMS_Road.56875
Resources
Statistics
Resources
AmountDumped
Done
Test

Figure 8-19 Expression Editor Used in the Creation of a Dynamic Road Segment

8.3.4 Support for Scripting

Users have the capability to bypass the GUI and create simulation models by writing VBA scripts. This is helpful in situations where large and repetitive modeling sequences are encountered. This function is performed through the script window shown in Figure 8-20. Again, the full Simphony object model is available to the user. The shown example illustrates how fifty copies of the "Batch" modeling element can be created with seven lines of code. This feature possible is made possible by the open architecture implementation of the Simphony system.



```
Function Script()
  Dim i As Integer
  Dim j As Integer
  For i=1 To 10
  For j=1 To 5
    Elements("56879").AddElement "Batch", i*100,j*100
  Next
  Next
End Function
```

Figure 8-20 Using the Scripting Capabilities to Create a Large Number of Elements

8.4 Summary

This chapter presented an overview of the Simphony environment and its components. Section 8.2 presented the development environment, including the base generic modeling element and its behaviors, the various services, and the Simphony Designer. Section 8.3 presented the Simphony Editor and its features.

The environment was developed as a unified modeling and simulation tool that supports the concepts presented in chapter 7. Special purpose modeling is directly supported through the developer's ability to create SPS templates. Graphical modeling is supported through the graphical modeling behavior of the elements and the extensive GUI service. Integrated modeling is supported through the database access and planning services. Modular and hierarchical concepts have been fully and thoroughly utilized through the relationship and hierarchical element behaviors. The hybrid tool development and utilization concept was supported through a controlled and simplified tool development approach and by providing users with the capability to create user elements.

Several manuals were written for Simphony users and developers. The Simphony Editor User's Guide and the Simphony Designer User's Guide provide information on how the two respective computer programs and their features work. The Simphony Developer's Guide is a comprehensive guide for the development of new SPS templates including several tutorials. The Simphony Reference Guide is a detailed listing and explanation of the Simphony object model, including all the supported language expressions. These manuals are available from the Construction Engineering and Management group at the University of Alberta (Hajjar 1999a).

Chapter 9 – Simphony Application Framework

9.1 Background

Discussions in previous chapters have concentrated on the fundamental construction modeling and simulation related concepts of this thesis. While this constitutes the majority of the contributions, there is one major concept from the software engineering and development arena that made it possible to transform the ideas into a practical computer system that can be deployed and tested. This concept is called object-oriented application frameworks and is presented here in order to provide a complete picture of the presented methodology including software design and implementation details. Further information on the internal implementation details of Simphony was documented as part of an internal department report (Hajjar 1999b).

9.2 Introduction

The majority of software development projects in the construction industry have followed the traditional approach. Applications are typically developed from scratch according to the requirements of the company. On some occasions, developers will reuse code from a previous project or take advantage of specialized third party components. This is referred to as "code-reuse" because only the programming code is being reused. While this practice reduces the development time, it requires the complete redesign of the application in every instance. Object-oriented application frameworks alleviate this limitation by capturing and reusing both the recurring application code and the design.

Object-oriented application frameworks, which will be referred to as application frameworks in the rest of this chapter, encapsulate a given design philosophy and a set of libraries that can be used to develop specific applications for a given domain. This means that overall development becomes a two step approach as illustrated in Figure 9-1.

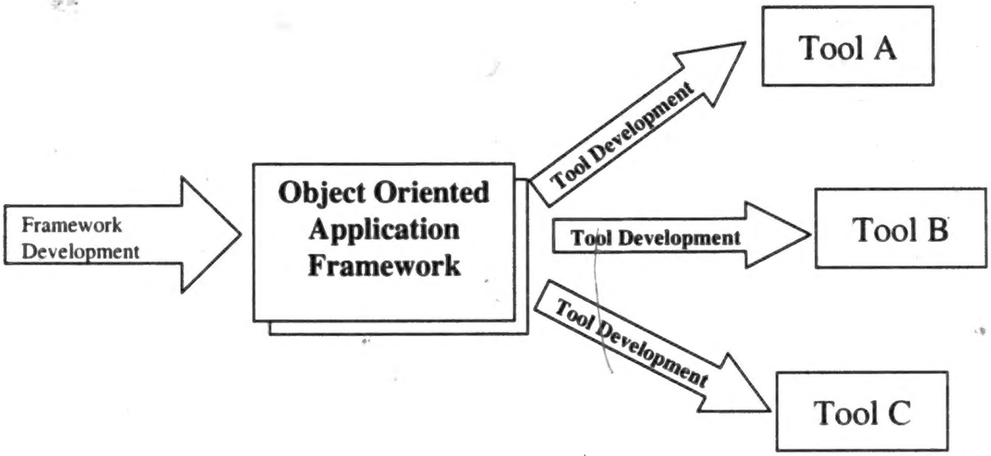

Framework Development
Object Oriented Application Framework
Tool Development
Tool Development
Tool Development
Tool A
Tool B
Tool C

Figure 9-1 Application Framework Approach to Tool Development

First, a number of expert developers who are familiar with a specific application domain develop the application framework. Examples of specific application domains are cost estimating systems, scheduling systems, and construction simulation systems. The developed application framework captures the broad and recurring business processes, transactions, and transformations within the target domain. Once the application framework is developed, specific tool implementations within the domain can be developed. For example, a specific implementation based on an estimating framework could include an earth-works estimating systems, a bridge estimating system, or a multi-story commercial building estimating system.

Understanding of application frameworks has greatly advanced in recent years. Section 9.3 will provide an overview of the basic concepts involved in framework design, implementation, documentation, and deployment. The information provided is based on application framework design and development guidelines as outlined in Froehlich, Hoover, Liu and Sorenson (1998). This work was also used as the basis for the design and development of the construction simulation application framework as will be explained in section 9.4 .

9.3 Overview of Application Framework Theory

9.3.1 Design

The first step in design of an application framework is the determination of its intended domain. This helps in defining the scope of the framework and the basic services that it should provide. Variability analysis is a technique that can be used to assist in this process. This involves a detailed analysis of features across a number of possible scenarios. Scenarios may include general descriptions, formal process models or. Ideally, actual tools developed for the intended domain. Variability analysis results in the identification of common and distinct features. The common features translate into the "frozen spots" of the framework. Frozen spots are framework components whose behavior tool developers have little or no control over. The identified distinct or variable features translate into the "hot spots" of the framework. Hot spots are places in the framework that tool developers can customize for specific requirements.

9.3.2 Implementation

Once the design of the framework is complete, the implementation can begin. Often, the process is iterative rather than linear; implementation may require the redesign of certain components, as the domain becomes more understood. During this process, the means by which developers will be customizing the framework are also determined.

9.3.3 Testing

Once the framework implementation is complete, it is tested through the implementation of several basic tools. Ideally, some of the tools originally used for the variance analysis are re-implemented. This proves the initial practicality and expressiveness of the developed application framework.

9.3.4 Deployment

Several issues are related to the deployment of application framework after their implementation and testing is completed. First, the framework must be introduced in a careful manner so that developers can gradually learn its basic requirements, features and components. This process can include roll out sessions, example applications, extensive documentation and tutorials. Second, the distribution means of the framework must be addressed. Possible options include source code or binary code distributions. Third, a process must be established to capture important feedback from the tool developers. This includes general support information, bug reports, and enhancement requests.

9.4 Application Frameworks for Construction Simulation

9.4.1 Overview

Chapter 7 explained how the unification of the various concepts was achieved through the formalization of a generic base modeling element and a set of development support services. This generic element can be used in the development of a special purpose simulation template that allows for modeling a specific construction domain. This approach to achieving unification naturally led to the use of an application framework as the basis for Simphony's development. As illustrated in Figure 9-2, the framework corresponds to the Simphony object model (SOM), which is used to build specific special purpose simulation (SPS) tools. These tools were previously referred to as SPS templates; they reside in the modeling element library.

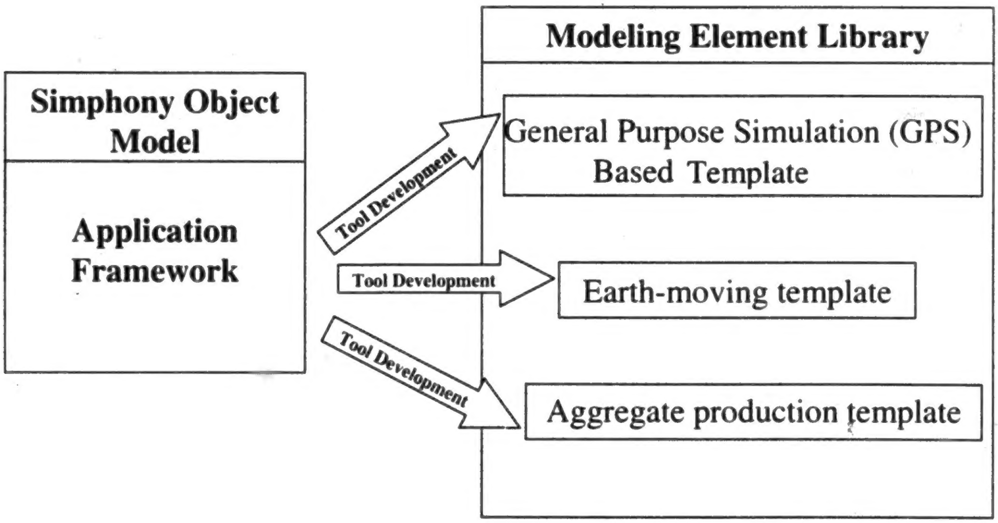

Simphony Object Model
Application Framework
Tool Development
Tool Development
Tool Development
Modeling Element Library
General Purpose Simulation (GPS) Based Template
Earth-moving template
Aggregate production template

Figure 9-2 Utilization of Simphony Application Framework for Generating SPS Templates

The Simphony object model encapsulates both the generic modeling element and all the required services. The benefit of this approach is that all the design and development knowledge common to all special purpose simulation tools is captured in a single place. This dramatically reduces the development complexity and duration for specific SPS tools and allows developers to concentrate on the core features of their intended domain. Further, by isolating the core component of the tool from the support elements, accurate comparisons across the various tools becomes possible. This result has allowed for a fair evaluation to be made across tools developed by students in a class assignment, as will be demonstrated in the next chapter.

9.4.2 Framework Design

9.4.2.1 Overview

The first stage in the design of an application framework is the determination of the target domain. SOM's domain is construction simulation tools. The domain is further limited to civil construction engineering. Further enhancements to the SOM may make it possible to support other types of construction areas but this has not been investigated as part of this thesis.

Domain analysis has been presented in the form of the unified modeling methodology previously found in Chapter 7. It was stated that the resultant tools must be based on such concepts as special purpose simulation modeling, graphical modeling, integrated modeling, and the hybrid approach to tool development and utilization. This realization came from an assessment of previous experiments with three domain applications: an earth-moving simulator (AP2-Earth), an aggregate plant production simulator (CRUISER) and a construction site dewatering simulator (CSD).

9.4.2.2 Variability Analysis

A variability analysis of the three applications revealed that a large proportion of features were identical in all tools; code had in fact being reused for these features. Further, of all the remaining features, the majority did not vary significantly and it was possible to generalize their functionality in order to standardize implementation. These common features became the "frozen spots" of the framework. The remaining features were genuinely unique to the specific tools. They were used in the identification of the "hot spots" of the framework.

Table 9-1 lists the features that were found to be the same across tools. Certain features existed in some tools but not in others. Table 9-2 lists the features that were, for the most part similar across tools. Some of the mentioned features did not exist in any of the tools but are required as stipulated by the unified modeling methodology. The fundamental basic variation across tools was the structure, or behavior, of the modeling elements. Although a standard list of broad behavior categories for modeling elements was identified, the individual definition of the behaviors across tools and even within specific tools varies considerably. Table 9-3 lists the identified behaviors and how they generally differ across modeling elements. Note that the features listed in all the tables correspond to a high level functional requirement and not necessarily a specific low level feature.

Table 9-1 Common Features across Construction Simulation Tools

Feature	AP2-Earth	CRUISER	CSD	Comments
Random Number Sampling	X	X		Both Ap2-Earth and CRUISER require a random number generation library able to transform results into a standard set of stochastic deviates. CSD is based on a deterministic model and does not required random sampling.
Project Plan Information Generation	X			AP2-Earth generates information based on the simulation run which can be used to construct a project plan in the form of an estimate or a schedule.
External database access	X	X		AP2-Earth and CRUISER can access an external database containing standard industry or company information.
Graphical user interface	X	X	X	All applications utilize the same graphical user interface library (Microsoft Windows GUI)
Modeling elements	X	X	X	All application employ a set of model building blocks, called modeling elements.

Table 9-2 List of Factored Features across Tools

Feature	AP2-Earth	CRUISER	CSD	Comments
Simulation Engine	X	X	X	CSD's simulation engine is a non-linear mathematical equation evaluator. CRUISER's is a steady state depth first analysis engine and Ap2-Earth's is a discrete-event engine. Each tool employs a different type of engine. However all three cases can be generalized to employ a discrete-event engine. Each tool would simply utilize the engine to a different extent.
Simulation progress and general feedback reporting	X	X		AP2-Earth generates a trace file of all the simulation events in the model and reports errors through the GUI. CRUISER generates information on the analysis processes at each element and reports errors through a central feedback manager. CSD does not require any simulation progress reporting but does inform users of errors directly through the GUI. Although the means of providing feedback and the required level of detail is different, all three tools can definitely benefit from a centralized feedback reporting mechanism.
Statistical Collection and	X	X		While CRUISER involves very simple statistical collection and analysis features, AP2-Earth's is more involved. It includes such analysis as

Analysis				intrinsic and non-intrinsic types. However, one is basically a subset of the other. This allows for a unified approach to statistical collection and analysis.
Simulation Model Persistence	X	X	X	All applications are able to store and retrieve the simulation models defined by the user. Ap2-Earth utilize a database approach in order facilitate data exchange while CSD and CRUISER employ a binary file approach. The unified modeling methodology stresses the importance of integrated modeling and as such, it is decided that all tools will utilize a database as the only means for model persistence.
Modeling Element Behaviors	X	X	X	While all tools require a set of modeling elements, the structure or behavior of each modeling element differs from one tool to another. A set of behaviors is factored from all three tools plus the further requirements of the unified modeling methodology to obtain a common set of behaviors with default functionality: attributes, relationships, hierarchy, simulation, integrity, statistics, planning, and graphical modeling. Most behaviors are defined differently for different modeling elements.

Table 9-3 List of Variable Features across Tools

Feature	AP2-Earth	CRUISER	CSD	Comments
Attributes	X	X	X	Each modeling element requires a unique set of attributes to represent its state. Some attributes allow users to provided parameter values at run-time, some provide simulation results and others are used for temporary and internal purposes. Attributes' internal representations also vary considerably between elements. Possible types include textual, numeric, and array based.
Relationships	X	X		While two of the tools require the notion of a relation between modeling elements, different modeling elements support different number and types of relationships.
Simulation	X	X	X	Modeling elements, across tools and within tools, define distinct simulation models in the form of discrete-event code handlers.
Integrity	X	X	X	Each modeling element requires a different set of rules that place constraints on its relationships, possible attribute values and other parameters.
Statistics	X	X		AP2-Earth and CRUISER localize the statistical collection and analysis at the modeling elements. Each modeling element generates a different number of statistics under different circumstances.

Planning	X			Planning information, according to the unified modeling methodology, will be derived from the individual implementation of the planning behavior of each modeling element.
Graphical Modeling	X	X	X	Each modeling element is represented graphically in a different manner in order to differentiate the different types of elements. Relationship information can also be represented differently depending on the tool or even the individual modeling element participating in the relationship. Modeling element attributes, which are manipulated by the user through the GUI, also employ different types of user interface controls. Some of controls utilized by the three tools include a text box, list box, calendar, and grid. Attributes designed for presenting results to the user also employ several graphical representations. For example, the site water table level attribute in CSD is represented by a three dimensional surface chart.

9.4.3 Framework Implementation

9.4.3.1 Overview

The presented domain variability analysis lead to the final design and implementation of the Simphony object model. The identified common features translated into the main framework services and a definition of the generic base modeling element. A simplified representation of the framework structure is shown in Figure 9-3. The previously identified common features were grouped into the services of the framework. The generic base modeling element encapsulates a default representation and can be used by developers to create custom simulation tools.

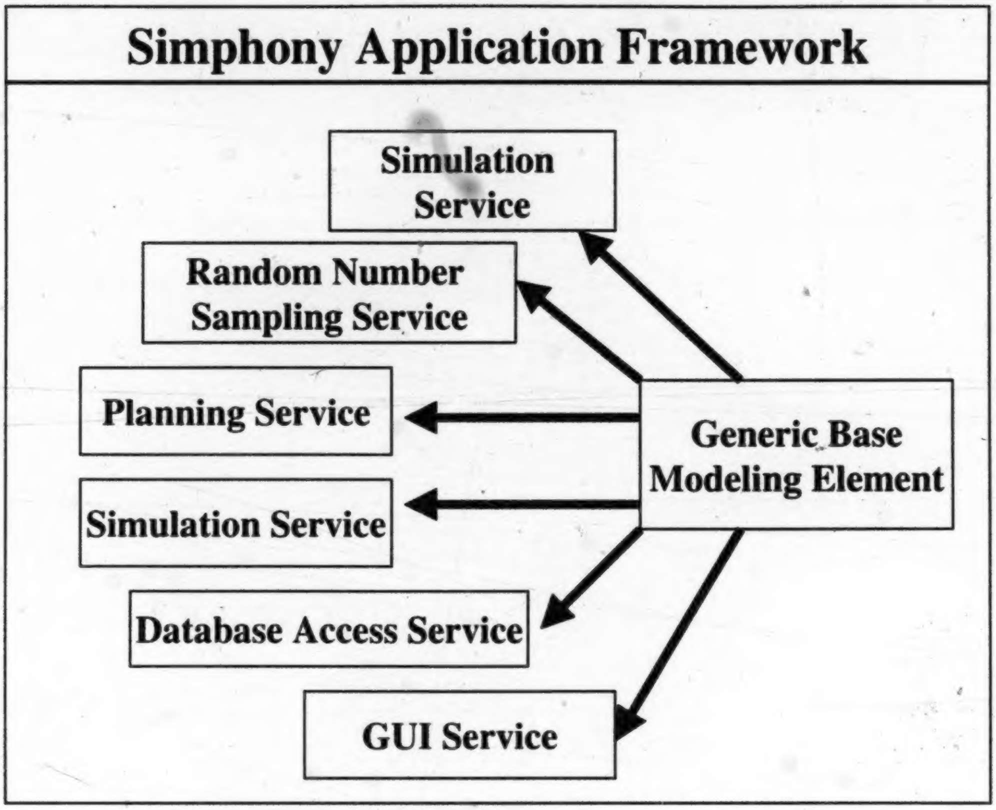

Simphony Application Framework
Simulation Service
Random Number Sampling Service
Planning Service
Simulation Service
Database Access Service
GUI Service
Generic Base Modeling Element

Figure 9-3 Simplified Representation of Simphony Application Framework

9.4.3.2 The Modeling Element as a Hot Spot

The generic base modeling element represents the main hot spot of the framework. This structure was implemented as a parameterized class, which developers can customize through the technique of composition. With composition, developers create objects based on the generic class and provide information as to the specific implementation of certain parameters.. The modeling element class parameters are the properties that can be manipulated through the overridable class methods. When based on this, construction simulation tool development is reduced to the process of declaring the set of required modeling elements and overriding the default implementation of the desired class methods. The detailed structure of the generic base modeling element class was presented in Section 8.2.2.

One way in which developers customize a given modeling element is related to its attribute table: an element that models a truck in an earth-moving simulation may contain an attribute table such as the one shown in Table 9-4.

Table 9-4 Sample Attribute Table for Truck Modeling Element

Name	Description	Internal Rep.	External Rep.	User Permissions	Min	Max
Type	Truck type	Text	List Box	Read Write		
Quantity	Number oftrucks	Numeric	Default	Read Write	1	100
Capacity	Truck capacity in cubic metres per hour	Numeric	Default	Read Write	5	200
Dumping time	Truck dumping duration	Distribution	Default	Read Write		
Ipriority	Truck priority at intersections	Numeric	Default	Read Write	1	
LPriority	Truck loading priority	Numeric	Default	Read Write		
Path	Path number to follow on branches	Numeric	Single	Read Write		

9.4.3.3 Concerns

The primary objective for developing the SOM was the simplification of the development process for new construction simulation tools. This objective, along with the unique nature of the framework domain, meant that a simplified tool development approach was needed. The result was the development of a custom integrated development environment (IDE) programming interface. This system is called the Simphony Designer and was previously presented in Section 8.2.4. The general idea is that developers use a special scripting language based on Visual Basic for Applications (VBA) to implement the modeling element class methods. The scripting environment exposes the framework services through special language statements. For example, the attribute table shown in Table 9-4 can be defined as part of the developer's implementation of the truck's OnCreate method which may appear as follows:

```
Public Function CEM_EMS_Truck_OnCreate(ob As CFCSim_ModelingElementInstance, _
                x As Single, y As Single) As Boolean

    CEM_EMS_Truck_OnCreate=True

    ob.AddAttribute "Type", "Truck Type", CFC_Text, CFC_ListBox, CFC_ReadWrite

    ob.AddAttribute "Quantity", "Number of Trucks", CFC_Numeric, CFC_Single, CFC_ReadWrite,1,100

    ob.AddAttribute "Capacity", "Truck Capacity in Cubic Metres", CFC_Numeric,_
                CFC_Single, CFC_ReadWrite,5,200

    ob.AddAttribute "DumpingTime", "Truck Dumping Duration", CFC_Distribution, _
                CFC_Single, CFC_ReadWrite

    ob.AddAttribute "IPriority", "Truck Priority at Intersections", CFC_Numeric, _
                CFC_Single, CFC_ReadWrite,1

    ob.AddAttribute "LPriority", "Truck Loading Priority ", CFC_Numeric, _
                CFC_Single, CFC_ReadWrite,1

    ob.AddAttribute "Path", "Path Number to Follow on Branches", CFC_Numeric, _
                CFC_Single, CFC_ReadWrite,1
End Function
```

The resultant scripts are evaluated at run-time during the tool execution process. Code syntax errors are highlighted during development while run-time errors are trapped and reported through the tracing service. This strategy resulted in a slightly different approach to traditional tool development. Developers, instead of creating new standalone tools, merely create new templates, in the form of customized modeling elements, to extend a single tool. This single tool has been previously presented as the Simphony Editor.

9.4.4 Framework Testing

The SOM was first tested through the re-development of AP2-Earth and CRUISER to use the framework. Results of these tests will be discussed in the next chapter. After the successful completion of these tools, a generic general-purpose simulation based tool was developed to evaluate the framework flexibility. This tool was presented in Section 8.2.7. This series of tests, performed by the framework developer, demonstrated the practicality and effectiveness of the framework.

9.4.5 Framework Deployment

Extensive user and reference documentation for the entire framework and the two supporting computer programs was written. The framework documentation included a complete reference manual of all the objects as well as a detailed developer guide that included several tutorial applications.

The framework was first introduced to graduate civil engineering students as part of a class module. The students had the benefit of one undergraduate programming course, one graduate computer applications course and basic computer simulation training using a commercial system. A series of four one-hour lectures and four two-hour lab sessions was given. The framework services and modeling element behaviors were presented along with some code illustrations. After the initial introduction, a series of tests, or case studies, was performed. These concentrated on evaluating the usability of the framework by novice developers. The students' first task was the development of a CYCLONE simulator as part of a class assignment,. Their second task was the development of a specialty simulation tool for a specific construction process of their choice for their course project. The code used to implement both the general-purpose simulation based tool and two re-developed tools were provided to students. The results of these case studies will be presented in the next chapter (Section 10.4). Bugs and requests for enhancements resulting from class testing followed a formal email reporting procedure.

This process resulted in several refinements to the framework in the form of new builds. The details of the build history are provided in Appendix 2.

9.5 Conclusions

This chapter discussed the application of object-oriented application framework concepts to the construction simulation domain. An application framework approach greatly reduces the development effort and allows for the standardization of the generated tools. Design of the framework resulted from a variability analysis of three previously developed standalone tools as well as the requirements of the unified modeling methodology.

Application frameworks represent the natural progression from archaic code reuse to development of reusable code libraries then to the development of structured object libraries. These were a critical factor in the implementation of ideas presented in this thesis. It is also strongly believed that they can be equally effective in other construction management domains.

Chapter 10 – Case Studies

10.1 Introduction

The presented unified modeling methodology, used as the basis for the development of Simphony, leads to improvements in two areas. First, simulation tools become more effective, powerful, and appealing to the average construction user. Second, the development process of these tools is dramatically simplified, and shortened.

The first statement has already been proven through the successful development, validation and deployment of the three tools presented in Chapters 3, 4, and 5. Although these projects were completed prior to the full development of the unified modeling methodology, their findings led directly to the formalization of the contributing concepts. As a result, the second area of improvement, related to tool development, is the primary focus of several case studies discussed in this chapter.

To test the hypothesis that development time is dramatically shortened, two of the previously presented tools, AP2-Earth and CRUISER, were re-developed using Simphony. The original development duration was then compared with the time required for re-development. The results of this case study are presented in Sections 10.2 and 10.3 . To test the other hypothesis that the development process is dramatically simplified, students from the Civil Engineering 606 simulation course undertook development of construction simulation tools using Simphony. This exercise is discussed in Section 10.4 .

10.2 AP2-Earth Redevelopment

10.2.1 Overview

A Simphony special purpose simulation template, called "CEM_EMS", was developed with most of the functionality provided by AP2-Earth. Users are able to model preparation, loading, hauling, dumping and spreading operations, and generate results in a similar manner to the AP2-Earth. The template code is provided in Appendix 3.

10.2.2 Earth-Moving Simulation using the CEM_EMS Template

The CEM_EMS template consists of sixteen modeling elements that can be used to define the various components of an earth-moving operation. Most of these elements have attributes that users can manipulate to modify the outcome of the simulation. A sample model based on this template is shown in Figure 10-1.

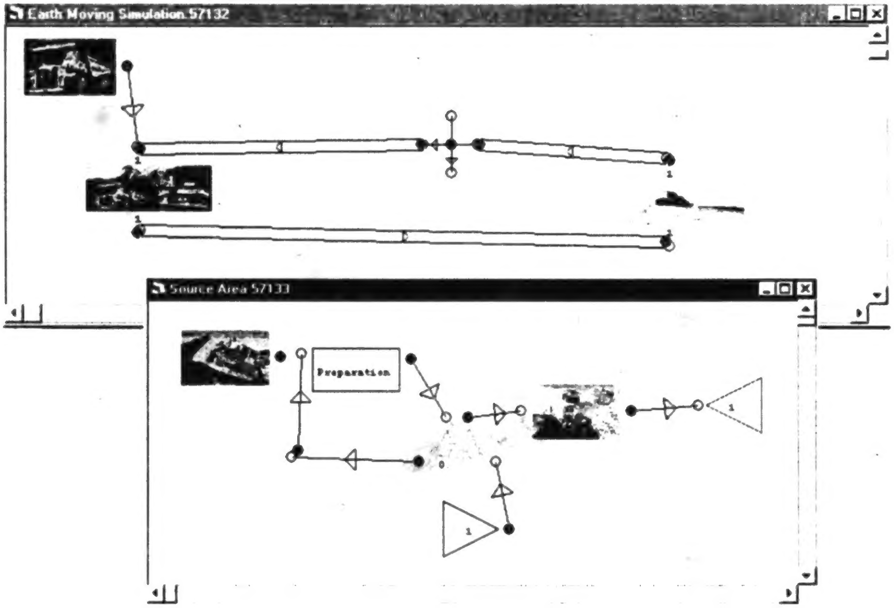


Figure 10-1 Sample Model Layout Based on the CEM_EMS Template

Models are built hierarchically in a top down fashion. Higher level elements generally represent overall operations with lower levels incorporating more details regarding the specific process. The available modeling elements, are illustrated hierarchically in Figure 10-2. At the highest level is the "CEM_EMS" element which corresponds to the template's name. Such root elements normally encapsulate high level model attributes, statistics, and shared resources. At the next level, seven elements can be used to define the source and placement areas, trucks, and road layout information, which is made up of road segments, intersections, branches and external traffic processes. Below the source area, represented by the "CEM_EMS_Source" element, a model can be defined using the six elements shown. These elements are related to the preparation and excavation processes. Below the placement area, represented by the "CEM_EMS_Placement" element, the spreading and dumping processes are defined using a set of seven modeling elements. As illustrated in Figure 10-2, five elements can be used for defining models inside both the source and the placement. The "CEM_EMS_TruckIn" and "CEM_EMS_TruckOut" elements are used for routing trucks from the parent (either source or placement) into the appropriate location in the sub-model.

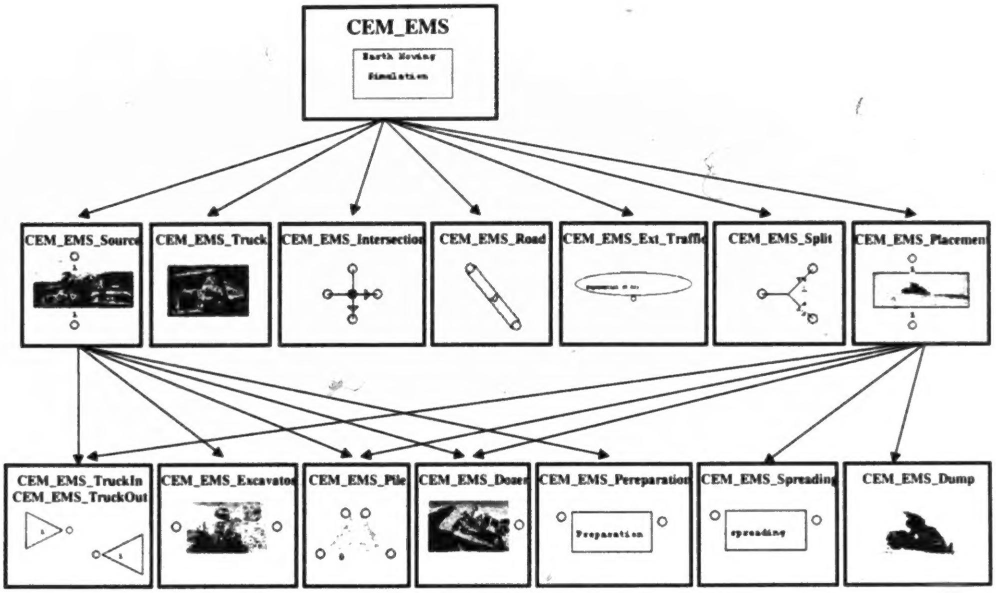

CEM_EMS
Earth Moving
Simulation
CEM_EMS_Sourc
CEM_EMS_Truck
CEM_EMS_Intersectio
CEM_EMS_Road
CEM_EMS_Ext_Traffi
CEM_EMS_Split
CEM_EMS_Placemen
CEM_EMS_TruckIn
CEM_EMS_TruckOut
CEM_EMS_Excavato
CEM_EMS_Pile
CEM_EMS_Doze
CEM_EMS_Pereparatio
Preparation
CEM_EMS_Spreadin
spreading
CEM_EMS_Dump

Figure 10-2 Modeling Elements of the CEM_EMS Template

Each modeling element provides a specific functionality and allows users to manipulate the internal behavior through its parameter attributes. Outputs and statistical analysis results are provided as needed for certain modeling elements. Table 10-1 lists the modeling elements, their functionality, parameters, outputs and any statistics they provide.

Table 10-1 Function of each CEM_EMS Template Element

Element	Function	Parameters	Outputs	Statistics
CEM_EMS	Serves as a simple container of a given earth-moving model.			
Source	Serves as a container of the preparation and the excavation/loading processes.	Amount To Haul	Amount Loaded	
Truck	Allows for the definition of one or more trucks and their properties.	Truck Type Number of Trucks Capacity Dumping Time Priority Path		Cycle Time
Intersection	Allows for the representation of	Expected Delay		Utilization

	traffic interaction amongst truck fleets.			Queue Length Waiting Time
Road	Models a one way single lane road segment.	Length Grade Rolling Resist.		
Ext_Traffic	Used to introduce an external traffic process into the model, typically at intersections.	Time of First Arr. Time between Arr. Priority		
Split	Used to route trucks to appropriate paths	Top Path Number Bottom Path Number		
Placement	Serves as a container of the dumping and spreading processes		Amount Dumped	
TruckIn	Used to route trucks from source or placement elements into sub-model.			
TruckOut	Used to route trucks from sub-model back to parent source or placement.			
Excavator	Allows for the definition of one or more excavators used for loading	Number of exc. Productivity		Utilization Queue Length Waiting Time
Pile	Models the interactions that occur at the loading or dumping material pile. Trucks will queue if not enough material has been prepared inside the source and dozer will queue if not enough material has been dumped inside the placement.	Starting Amount	Current Amount	File Length Waiting Time
Dozer	Allows for the definition for one or more dozer used for preparation and spreading operations.	Number of dozers Capacity Productivity		
Preparation	Used to model preparation activity inside a source	Amount to Prepare	Amount Prepared	
Spreading	Used to model the spreading operation inside a placement		Amount Spread	
Dump	Used to model a dumping location inside a placement	Maximum allowed trucks at any time	Amount Dumped	Utilization Queue Length Waiting Time

10.2.3 Strategies for CEM_EMS Template Development

During the development of the template, it was determined that Simphony's hierarchical modeling support should be employed in order to guide users in building models using a "top-down" approach. At the second level of the hierarchy, below the root element, the model would correspond to what a planner would initially "jot down" on paper to explain the project to another person. At the next level, as part of the source and placement sub-models, the preparation, excavation, dumping and spreading operations would be defined. For defining the simulation behavior of the preparation and spreading elements, the GPS (general purpose modeling) template was used as an alternative to hard-coding the discrete-event code inside each modeling element. The benefits of this approach are that development time is reduced as coding is not required and the user, although not required to, is able to modify the underlying simulation behavior by defining an alternative sub-model using the GPS template. This means that a fourth level in the hierarchy exists below the preparation and the spreading operations. It is based on modeling elements from the GPS template as illustrated in Figure 10-3.

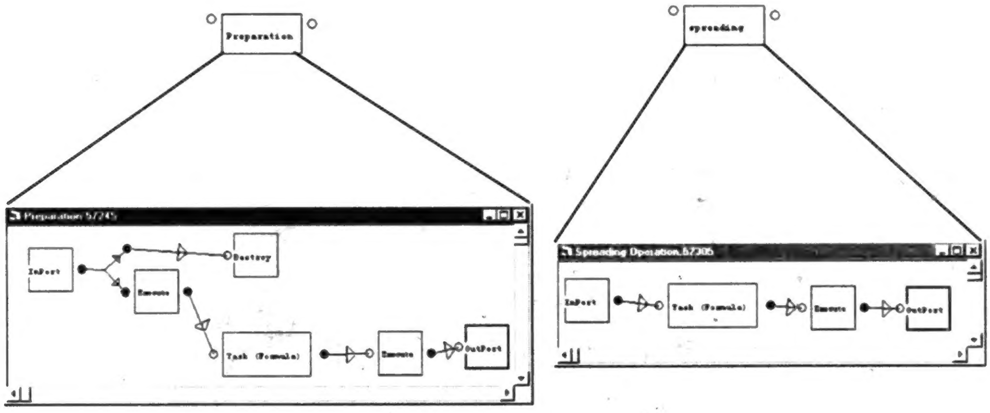

Preparation
spreading
Preparation 57245
InPort
Destroy
Execute
Task (Formula)
Execute
OutPort
Spreading Operation.57305
InPort
Task (Formula)
Execute
OutPort

Figure 10-3 Definition of Simulation Behavior through GPS Template

10.2.4 Enhancement Over the Original Tool

The use of hierarchy and modularity concepts for the implementation of the CEM_EMS template resulted in a more flexible and extendable tool. Flexibility was gained by exposing more of the internal implementation details of certain elements. For example, with AP2-Earth, the user had limited control over the behavior of the preparation and excavation processes. With the developed template, users can view the definition of these processes and modify them in any desired manner. The tool can also be extended by users who are familiar with the GPS template. As was explained earlier, the definition of the spreading and preparation operations can be modified. The same approach can be followed in combining the elements based on the CEM_EMS template with other elements based on the GPS template for modeling certain scenarios. An illustration of how this can be done was provided in Figure 7-9.

Another advantage of the Simphony template is that users are able to store a given portion of their model in the user library for use in other projects.

10.2.5 Comparison of Development Efforts

Analysis of historical records showed that the original development of the AP2-Earth project was equivalent to a one person labor year. This involved time spent learning the fundamentals of earth-moving operations, site visits, company interviews, site deployment, documentation and integration. The actual design and programming time, which involved C++ coding, database development, and report design and construction, was approximately 400 hours. This value can be compared with the Simphony template development time, which is detailed in Table 10-2. The total duration was 28 hours.

Table 10-2 Development Hours for the CEM_EMS Template Components

Task	Hours
Overall Design	5
Source	2.5
Placement	2.5
Road	3
TrucksIn	1
TrucksOut	1
Dump	2
Excavator	2.5
Pile	3.5

Spreading	2.5
Preparation	2.5

10.3 CRUISER Redevelopment

10.3.1 Overview

In a similar manner, a Simphony template, called "CEM_CRUSH", was developed with the same functionality as CRUISER. A sample model layout based on this template is shown in Figure 10-4. The Simphony code for this template is not provided due to its length. However, it has been published as part of a department internal report (Hajjar 1999c).

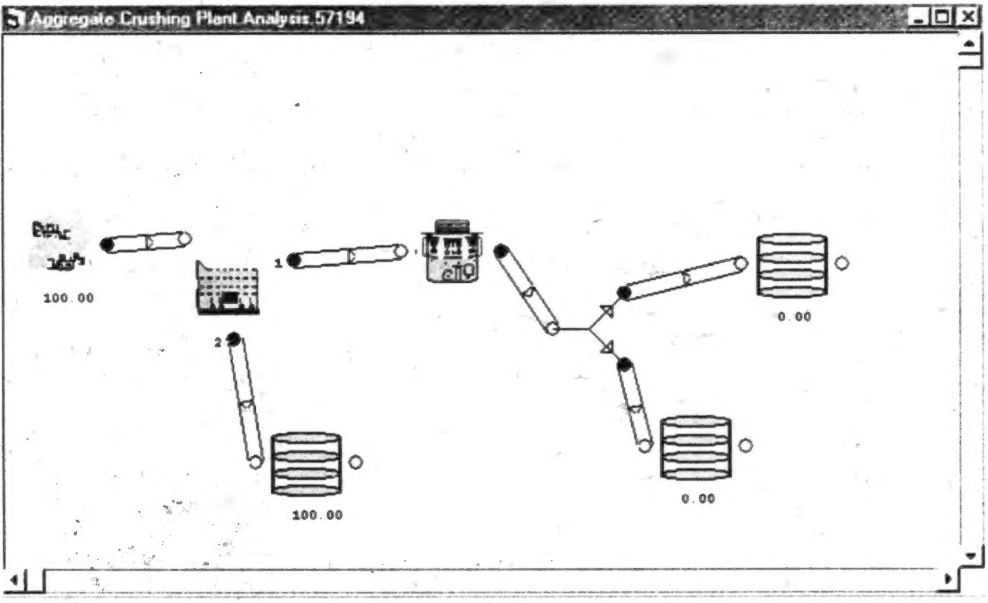

Aggregate Crushing Plant Analysis.57194
100.00
1
2
0.00
0.00
100.00

Figure 10-4 Sample Model Layout Based on the CEM_CRUSH Template

10.3.2 Aggregate Plant Simulation Using the CEM_CRUSH Template

The set of modeling elements making up the CEM_CRUSH template are similar to the corresponding elements of CRUISER in both representation and functionality. The one major difference is that some elements take advantage of hierarchy and modularity concepts. The hierarchical relationships of the modeling elements are illustrated in Figure 10-5.

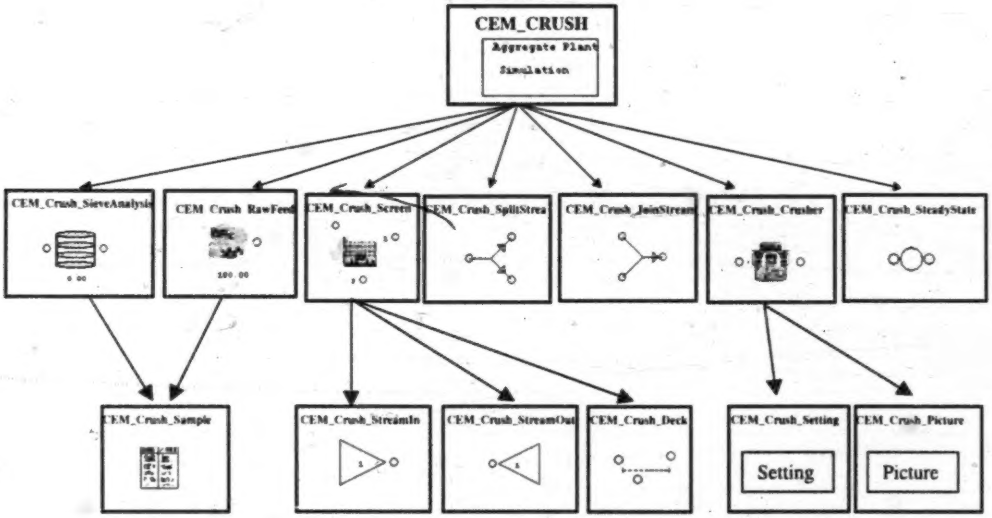

CEM_CRUSH
Aggregate Plant Simulation
CEM_Crush_SieveAnalysis
0.00
CEM Crush RawFeed
100.00
CEM_Crush_Screen
1
2
EM_Crush_SplitStrea
CEM_Crush_JoinStream
CEM_Crush_Crusher
CEM_Crush_SteadyState
CEM_Crush_Sample
CEM_Crush_StreamIn
1
CEM_Crush_StreamOut
1
CEM_Crush_Deck
CEM_Crush_Setting
Setting
CEM_Crush_Picture
Picture

Figure 10-5 CEM_CRUSH Template Structure

The raw feed element (CEM_Crush_RawFeed) is used to define the properties of the input raw stream into the plant. The feed rate is manipulated through the attribute form. The actual gradation of the samples is provided by defining one or more sample elements (CEM_Crush_Sample) as children of the raw pile element. Each defined sample represents a single observation.

The sieve analysis element (CEM_Crush_SieveAnalysis) is used at strategic locations throughout the plant model in order to collect observations. The element will analyze incoming stream passing through it and produce a gradation graph similar to the one shown in Figure 4-6. This functionality was previously a property of the conveyor element; it was transferred to allow users to explicitly determine the places where analysis should take place. This eliminates unnecessary calculations. The sieve analysis modeling element also replaces the functionality of CRUISER's product pile. The product pile was used to examine the generated gradation along with a user defined envelop representing the desired gradation. To generate this type of output (shown in Figure 4-3), two sample elements (CEM_Crush_Sample) can be added as children of the sieve analysis element. One represents the upper bound and the other represents the lower bound. If this is done, the sieve analysis element will simply add the streams represented by the sample elements to the gradation graph along with the results of the analysis, resulting in the desired type of output. Any number of child samples can be added to compare the analysis results with any reference gradation.

The screen element (CEM_Crush_Screen) is used for the modeling of the size separation process. Several other support elements are also used in defining the various size separation parameters. The deck (CEM_Crush_Deck), StreamIn (CEM_Crush_StreamIn) and StreamOut (CEM_Crush_SteamOut) elements are used to define a sub-model inside the screen element that determines the exact stream flow across the various decks and splits of the screen. For example, a simple single-deck screen can be represented with the model shown in Figure 10-6A while a double-deck screen with a split top deck can be represented with the model shown in Figure 10-6B.

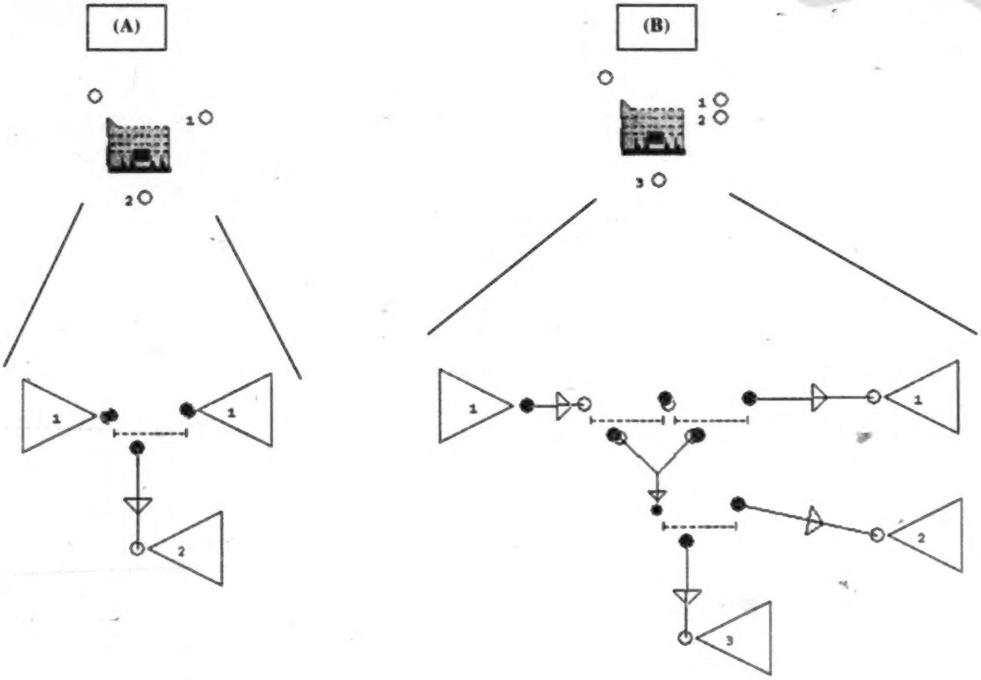


Figure 10-6 Modeling Size Separation Processes Using Screen Sub-Models

The crusher (CEM_Crush_Crusher), setting (CEM_Crush_Setting), and picture (CEM_Crush_Picture) elements are used in the modeling of the size reduction process. Size reduction modeling, as explained in Section 4.3.7, depends on empirical tables that provide the expected gradation based on a given crusher setting. A crusher is defined by first creating a crusher element and then defining the empirical table as a sub-model. Each possible crusher setting, along with its expected gradation, is defined using a single setting element. A typical crusher could contain up to ten child setting elements representing the ten possible crusher settings. The picture element is used to customize the graphical representation of the crusher element. The described modeling process, involving the definition of multiple setting child elements and their corresponding expected gradation, must be defined only once for each type of crusher. Once the new crusher type is defined, it can be stored in the user element library for later use.

10.3.3 Strategies for CEM_EMS Template Development

The re-development of CRUISER involved a further analysis of the basic required data structures. This led to the factoring of several components in order to reduce the amount of coding and resulted in better use of object-oriented modeling concepts. The sample element is one example of this factoring. In CRUISER, sample related data structures were duplicated in the raw feed, desired gradation and product pile elements. Although these elements manipulated the samples using shared routines, certain aspects, including the user interface, were individually implemented. The sample element, once defined, is used whenever gradation data was required from the user. For example, it is used both to obtain raw feed sample gradation for the raw feed element and to define desired gradations inside the sieve analysis element.

The hierarchical concepts used in Simphony made it possible to obtain complex data from the user. For example, the crusher element required the definition of an empirical gradation table consisting of a custom two-dimensional structure. This was done by representing each column of the table with the setting element, which, in turn, allows the user to define the expected gradation for the corresponding setting. During simulation validation, the crusher element first ensures the existence of a child setting element that defines the expected gradation for the user specified crusher setting. If an element is found, the crusher element stores a reference to the element for alter use. The code used to do this is as follows:

```
Public Function CEM_Crush_Crusher_OnCheckIntegrity(ob As CFCSim_ModelingElementInstance) As
Boolean
```

```
    Dim child As CFCSim_ModelingElementInstance
    Set ob("SettingObject").Reference =Nothing

    For Each child In ob.ChildElements
        If child.ElementType="CEM_Crush_Setting" Then
            If ob("Setting")=child("SettingLabel") Then
                CEM_Crush_Crusher_OnCheckIntegrity=True
                Set ob("SettingObject").Reference = child
                Exit Function
            End If
        End If
    Next

    Tracer.Trace "Current crusher setting is invalid. ","Integrity"
    CEM_Crush_Crusher_OnCheckIntegrity=False
End Function
```

During the simulation, when a stream entity reaches a given crusher, the crusher element uses the stored setting element reference to obtain the expected gradation information and use it as part of the processing algorithm.

The screen element was also designed to take advantage of hierarchy and modularity support. The basic processes involved in modeling a screen were separated and implemented as individual elements. This resulted in a the use of a screen sub-model to define specific stream flow across decks and splits. The StreamIn element is used to route the incoming stream entities to the appropriate deck. Multiple decks are then used to represent the various decks and splits of a given screen. The StreamOut element is finally used to direct stream entities back to the higher level model.

10.3.4 Enhancements Over the Original Tool

Many benefits were gained from the redevelopment of CRUISER as a Simphony template; some of these were mentioned in the previous section. Other benefits include the ability to extend the model with GPS-based elements and to combine models based on the CEM_Crush template with elements based on other templates. Some of the more specific advantages include inherent support for custom crushers, extended flexibility of size separation process modeling, and storage of commonly used models in the user element library.

CRUISER did allow for custom crushers to be defined, but that involved using a separate program to define the empirical tables. By designing and implementing the crusher element using the approach explained in the previous section, it becomes possible to define custom crushers as part of the modeling environment.

Users are also given more control over the modeling of the size separation process. Whereas before all aspects of this process were controlled through the screen element parameters, the current approach exposes the user to the inner workings of the screen element and allows them to customize it as desired.

The user element library can be used to greatly simplify the modeling process for aggregate production operations. Commonly occurring sample gradations can be stored by creating a sample element, defining the gradation, and storing the result in the library for later use in other projects. The same applies to commonly used crushers. Aggregate producers will typically posses a limited number of crushers. As a result, they can define the properties for each crusher element and store the information in the library. Future simulation projects can access the library and use the exact crusher which is to be used on the planned quarry. The same idea applied to screens; commonly used screen setups, such as single deck, double deck, and single deck with split top, can be modeled once, stored in the library and later accessed for use in other projects.

10.3.5 Comparison of Development Efforts

Historical records revealed that total project investment was equivalent to one and a half person years, of which 650 hours was actual design and programming time. The remaining time was spent on process discovery, documentation, and implementation. The development of the equivalent Simphony template, detailed in Table 10-3, took approximately 40 hours.

Table 10-3 Development Hours for the CEM_CRUSH Template Components

Task	Hours
Overall Design	5
Raw Pile	3
Sample	3
SieveAnalysis	8
AddStream	3
SplitStream	3
Crusher	9
Screen	4
Deck	2

10.4 Class Experiments

10.4.1 Overview

Simphony was introduced to five students in a graduate class. Each student had the benefit of one computer applications course, a basic Visual Basic education, and simulation training in the course. After attending four one-hour lectures and four two-hour lab sessions, students were asked to develop a Simphony CYCLONE template as part of an assignment. CYCLONE is an activity based simulation system that utilizes a set of five basic modeling elements. A sample model was illustrated in Figure 7-2. The students' second task was to develop a template for either paving or tunneling construction processes as part of their course project.

10.4.2 CYCLONE Template

CYCLONE was selected for the assignment because it consisted of a relatively small and easily understood number of modeling elements. Further, a standalone CYCLONE simulation tool was developed by a third-year computing science student for instructional purposes. The programmer's development time was approximately 230 hours. This did not include time spent becoming familiar with the CYCLONE method. The template development time of the students, which averaged approximately 40 hours, is detailed in Table 10-4.

Table 10-4 Summary of Students' CYCLONE Template Development

Student	Hours	Grade	# of Lines of Code	Normalized Hours
A	35	60%	256	58.3
B	36	100%	418	36
C	15	90%	325	16.7
D	26	60%	255	43.3
E	32	65%	260	49.2

Provided hours were first normalized by dividing them by the assigned percentage grade. This was done as a means of extrapolating the number of hours it would have taken to provide a fully functional tool. Some students did not implement certain features such as priority queue allocations and counter statistics. The number of lines of code is shown in order to illustrate the relative complexity of the template. This number can be compared to the "CEM_EMS" template, which consists of 1168 lines, or the "CEM_CRUSH" template, which consists of 2059 lines.

10.4.3 Project

Students were instructed to design and develop a Simphony template to allow for the modeling of either paving or tunneling operations. All students were able to successfully complete the basic project requirements. The degree of flexibility and effectiveness of the resulting tools varied between students. The results of the projects are summarized in Table 10-5. Development time averaged 41 hours for the paving template and 67 hours for the tunneling template. A sample model based on a paving template from student B is shown in Figure 10-7 and a sample model based on the tunneling template from student E is shown in Figure 10-8.

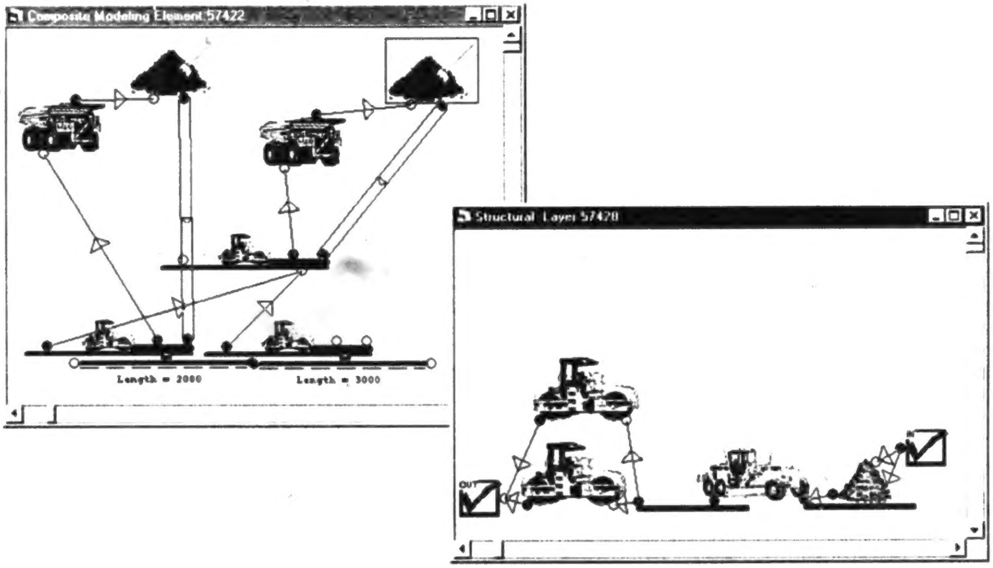

Composite Modeling Element 57422
Length = 2000
Length = 3000
Structural Layer 57420
OUT

Figure 10-7 Sample Model Layout Based on Student B's Paving Template

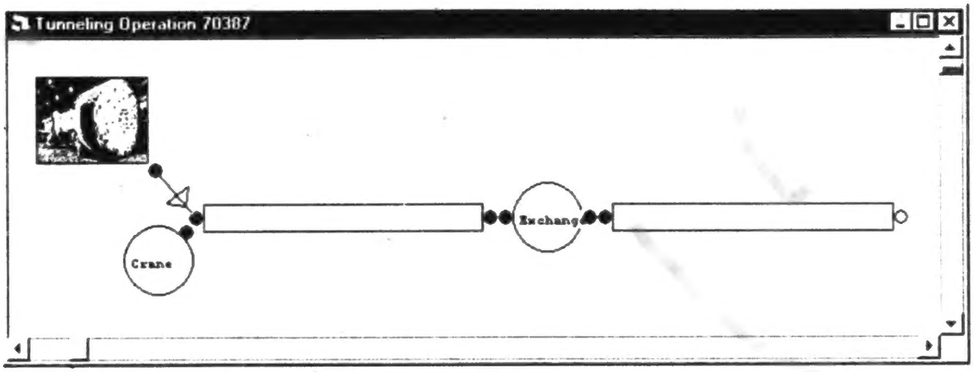


Figure 10-8 Sample Model Layout Based on Student E's Tunneling Template

Table 10-5 Summary of Students' Project Template Developments

Template	Student	Hours	Grade	# of Lines of Code	Normalized Hours
Paving	A	25	60%	484	41.7
Paving	B	35	100%	1161	35
Paving	C	38	80%	705	47.5
Tunneling	D	47	70%	722	67
Tunneling	E	40	60%	300	66.7

10.5 Summary

This chapter discussed several case studies that demonstrated that Simphony can lead to a dramatic simplification and shortening of the development effort for new construction simulation tools. Two case studies, which involved the re-development of AP2-Earth and CRUISER, showed how development time was reduced by a factor of fifteen. Experiments involving graduate students proved that novice developers are able to easily develop new simulation tools. Further, the time required for students to develop a Simphony-based CYCLONE tool was shorter than the time it took a programmer to develop a standalone version by a factor of six.

Chapter 11 – Final Discussion

11.1 Research Summary

Research presented in this thesis began with the goal of transferring simulation technology into the construction industry and making simulation based tools as common as estimating and scheduling systems. In order to achieve this goal, it was determined that a study would have to be done first to identify the basic requirements that simulation tools must possess in order to be successful.

This study was the focus of the first phase of research. It resulted in three custom simulation tools for earth-moving, aggregate production and site dewatering operations. These tools were developed and refined based on the requirements of leading local contractors with expertise in their respective industries. An analysis of the success factors of the tools as well as their limitations led to the identification of a set of features that all simulation tools must support. These features were grouped into several categories and can be summarized as follows:

1. The user interface should support graphical representation and manipulation of the model structure. Integrity violations should be trapped as soon as possible and reported to the user through the modeling interface in a helpful fashion. Graphical modeling support should be the primary means of model definition and manipulation. However, advanced users should still be accommodated and allowed to bypass the graphical system.
2. The modeling process should be done in a manner that is natural and relevant to the specific target domain of the simulation tool. Users should not be exposed to the

abstract underlying constructs, which require expertise with fundamental simulation concepts.

3. Construction methods vary in complexity and as a result the simulation tools must be able to accommodate different types of simulation processors.
4. Results generated by the simulation tools should be of immediate relevance to the target user. Specific post simulation analysis should be performed when required with results presented in a familiar and natural manner.
5. Simulation tools must be able to integrate with existing systems in order to reduce the data entry requirements as well as to generate information for use by existing systems. The generated information from each tool should follow a standard structure in order to simplify its analysis process by external systems.
6. Users should be able to combine models based on several tools in an effective manner in order to allow for the modeling of complete projects involving multiple construction methods.
7. Tools should support and even encourage the reusability of exiting simulation models.
8. Tool developers should be able to create new tools in a relatively short time with minimal effort. The tool development process should be standardized in order to provide inherent support for all mentioned features.

Based on these identified features, a set of concepts was then formulated. Special purpose simulation modeling allowed for the development of intuitive, domain-based modeling elements. Graphical modeling allowed for the simplification of the modeling process for novice users. Integrated modeling defined how information can be obtained from other systems through a relational database management system and introduced a standard for the generation of project planning type of data. Modular and hierarchical modeling allowed for the modeling of large simulation projects, linking of models based on different SPS tools, and empowerment of users with development capabilities. The hybrid tool development and utilization approach explained how it is more beneficial for a simulation system to support a spectrum of developers and users with different skills and how their needs can be satisfied. This set of formulated concepts was then combined into a unified modeling methodology. The unification process was supported with the concept of object-oriented modeling.

A computer system called Simphony was then developed based on the unified modeling methodology. Simphony allows for the creation of new SPS tools in the form of modeling element templates. Development of new tools is greatly simplified and shortened as developers create new tools by inheriting the pre-defined functionality of a generic modeling element and then customizing it as required. The development process is supported by a set of services that provide commonly used routines for such tasks as discrete-event simulation processing and statistical collection and analysis.

The design and implementation of the Simphony system was guided by the principles of object-oriented application frameworks. This is a concept from the software engineering discipline that provides guidelines on designing and implementing libraries that capture a recurring software solution within a given application domain.

Several case studies were performed in order to test the level of tool development simplification. Through a re-development of the earth-moving and aggregate production simulation tools, it was discovered that the simplification factor can be as high as sixteen for developers who are familiar with Simphony. It was further demonstrated that novice developers are able to produce new Simphony-based tools at a pace that exceeds that of a programmer using a commercial development system by a factor of six.

11.2 Summary of Research Contributions

The described research has led to numerous contributions to construction simulation research. During the first phase of research, actual simulation knowledge was transferred to local construction companies through the developed tools. Collaborating companies used the tools as part of their decision making process. Further, a set of features that ensure the success of specialized construction simulation tools was identified. As part of the second phase, several concepts were formalized and combined to form a unified modeling methodology. This methodology was used to develop a fully functional computer system, which dramatically simplified and shortened the development time of new construction simulation tools.

These represent some of the concrete contributions that resulted from this research. There are numerous other anticipated contributions. By standardizing the structure of simulation tools through the transformation of their definition process into a declarative format, a standard representation of the fundamental construction process emerges. This isolation of core tool requirements from the support elements could potentially allow for the automated analysis of individual construction processes as well as the representative comparison of different processes.

Further, the centralized and standardized approach to the representation of the construction process and project data leads to many contributions. The modeling element library, which is where all the specific tool definitions are stored, is, in effect, a formal representation of the construction method within a company or across the industry. The project data captures planning decisions not only regarding the overall project breakdown, high level relationships and resources, but also regarding the underlying reasoning process of the planner.

11.3 Recommendation for Future Development

The presented research demonstrates how the unified modeling methodology can be followed to build effective and practical simulation tools. This was done through the redevelopment of earth-moving and aggregate production templates based on the original tools which were successful in an industry setting. It is recommended that this horizontal type of research expansion be continued through the development of new templates for common construction methods; thus leading to a complete process library.

It is also recommended that a vertical expansion of the system be done in order to improve the functionality of Simphony. This is inherently supported through the presented user service concept. Specific recommendations for new services, implemented as ActiveX libraries, include an optimization library and an artificial neural network training and recall support library.

Bibliography

AbouRizk, S., Hajjar, D. (1998). "A Framework for Applying Simulation in the Construction Industry." Canadian Journal of Civil Engineering, 25(3), 604-617.

Ahl, O.J., Nygaard, K. (1966) "Simula - An Algol-Based Simulation Language." Communications of the ACM, 9, 671-678.

American Society For Testing Materials (1996). "Mechanical Size Analysis of Extracted Aggregate." ASTM test designation D5444-94.

Aziz, N.M., Burati, J.L., and Ozodigwe, D. (1989). "A Microcomputer Model for Dewatering Construction Sites." Proceedings of the Seventh National Conference on Microcomputers in Civil Engineering, 201-205.

Ball, P., Love,D. (1995). "The Key to Object-Oriented Simulation: Separating the User and the Developer", Proceedings of the 1995 Winter Simulation Conference, Arlington, VA, USA.

Bischak, D.P., Roberts, S.D. (1991). "Object-Oriented Simulation." Proceedings of the 1991 Winter Simulation Conference.

Cederapids (1984b). Pocket Reference Book 4th Edition, Iowa Manufacturing Company, Cedarapids Inc,, Cedar Rapids, USA.

Chang, D.Y. and Carr, R.I. (1987). "RESQUE: A Resource Oriented Simulation System for Multiple Resource Constrained Processes." Proceedings of the 1987 PMI Seminar/Symposium, Milwaukee, Wisconsin, 4-19.

Chang, D.Y (1991). "Object-Oriented Simulation System for Construction Process Planning." Construction Congress 91, ASCE, New York, NY, USA, 626-631.

Chehayeb, N. (1997). "CRUISER Implementation Results", Internal Report, Department of Civil Engineering, University of Alberta, Edmonton, Alberta.

Cubert, R.M., Goktekin, T., Fishwick, P.A. (1997). "MOOSE: architecture of an object-oriented multimodeling simulation system." Proceedings of Enabling Technology for Simulation Science, Part of SPIE AeroSense '97 Conference, Orlando, Florida, 22-24.

Fritz. D.,G., Sargent, R.G., Daum, T. "Overview of HI-MASS (Hierarchical Modeling And Simulation System)", Proceedings of the 1995 Winter Simulation Conference, WSC 95. Arlington, VA, USA.

Froehlich, G., Hoover, J., Liu, L., Sorenson, P. Designing Object-Oriented Frameworks, in CRC Handbook of Object Technology, CRC Press, 1998, in press.

Hajjar, D. 1999a, "Simphony Documentation", Internal Report, Construction Engineering and Management, Department of Civil Engineering, University of Alberta.

Hajjar, D. 1999b, "Simphony Implementation", Internal Report, Construction Engineering and Management, Department of Civil Engineering, University of Alberta.

Hajjar, D. 1999c, "Simphony Code for the CEM_EMS and CEM_CRUSH Templates", Internal Report, Construction Engineering and Management, Department of Civil Engineering, University of Alberta.

Hajjar, D., AbouRizk, S., (1996). "Building a Special Purpose Simulation Tool for Earth-Moving Operations." Proceedings of the 1996 Winter Simulation Conference, ASCE, 1313-1320.

Hajjar, D., AbouRizk, S. (1998). "Modeling and Analysis of Aggregate Production Operations." Journal of Construction Engineering and Management, ASCE, 124(5).

Hajjar, D., AbouRizk, S., Xu, J. (1998). "Optimizing Construction Site Dewatering Operations using CSD." Canadian Journal of Civil Engineering, CSCE, 25(3).

Hajjar, D. AbouRizk, S.M. , Mather, K. (1999). "Integrating Neural Networks with Special Purpose Simulation." Proceedings of the 1998 Winter Simulation Conference.

Halpin, D. W. (1977). "CYCLONE: Method for Modeling of Job Site Processes" Journal of the Construction Division, ASCE, 103(3),489-499.

Hancher, D. E. and Havers, J. A. (1972). Mathematical Model of Aggregate Plant Production, ASCE, New York, USA.

Huang, R., A.M Grigoriadis, and D. W. Halpin (1994). "Simulation of Cable-stayed Bridges Using DISCO." Proceedings of Winter Simulation Conference, 1130-1136.

Ioannou, P.G. (1989). "UM-CYCLONE User's Guide." Department of Civil Engineering, The University of Michigan, Ann Arbor, Michigan.

Johnston,D. W. (1981). "Linear Scheduling Method for Project Planning Analysis." Journal of Construction Engineering and Management. ASCE. 107(2), 247-261.

Karhl, D. (1995). "Building End User Applications With Extend." Proceedings of the 1995 Winter Simulation Conference, Arlington, VA, USA

Kreutzer, W. (1986). System Simulation Programming Styles and Langauges. Addison-Wesley Publishing, Don Mills, Ontario.

Lasdon, L.S., Waren., A.D., Jain, A., and Ratner, M. (1978). "Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming." ACM Transactions on Mathematical Software, 4(1),34-50.

Liu, L. Y. and P. G. Ioannou (1992). "Graphical Object-Oriented Discrete-Event Simulation System." Proceedings of Winter Simulation Conference, ASCE, 1285-1291.

Liu, L. Y. and P. G. Ioannou (1993). "Graphical resource-based object-oriented simulation for construction process planning." Proceedings of the 5th International Conference on Computing in Civil and Building Engineering - V-ICCCBE. Anaheim, CA, USA

Luna, J. (1993). "Hierarchical Relations in Simulation Models." Proceedings of the 1993 Winter Simulation Conference.

Martinez, J. and P. G. Ioannou (1994). "General Purpose Simulation with Stroboscope." Proceedings of Winter Simulation Conference, ASCE, 1159-1166.

Mathewson, S.C. (1989). "Simulation Support Environments." Computer Modeling for Discrete Event Simulation, ed. M. Pidd, 57-100. London: J. Wiley and Sons.

McDonald, M.G., and Harbaugh, A.W., (1988). "A Modular Three-Dimensional Finite-Difference Ground-Water Flow Model", United States Geological Survey, 437 National Center Reston, VA 20192.

McKim, C.S., Matthews, M.T. (1996). "Modular Modeling System Model Builder", Proceedings of the 1996 31st Intersociety Energy Conversion Engineering Conference, IECEC 96. Part 3 (of 4). Washington, DC, USA.

Muskat, M. (1953). The Flow of Homogeneous Fluids Though Porous Media, McGraw-Hill, New York.

Oloufa, A.A. (1993). "Modeling Operational Activities in Object-Oriented Simulation." Journal of Computing in Civil Engineering., ASCE, 7(1), 94-106.

Oloufa, A.A. (1994). "User-oriented approach to construction simulation of buildings." Microcomputers-in-Civil-Engineering, 9(6), 1994, 425-433.

Paulson, B.C. Jr., (1978). "Interactive Graphics for Simulating Construction Operations." Journal of the Construction Division, ASCE, 104(1),69-76.

Paulson, B.C., Jr., Chan, W.T., Koo, C.C. (1987). "Construction Operation Simulation by Microcomputer." Journal of Construction Engineering and Management, ASCE, 113(2), 302-314.

Peurifoy, R.L., Ledbetter, W.B., and Schexnayder,C.J. (1996). Construction Planning, Equipment and Methods. McGraw-Hill, New York, USA.

Pidd, M. (1992). "Guidelines For the Design of Data Driven Generic Simulators for Specific Domains." Simulation, 59 (4),237-243.

Pioneer (1988). Fact and Figures. (80-81) Portec Pioneer Division, Yankton, USA.

Powers, J.P. (1981). Construction Dewatering. John Wiley & Sons, New York, N.Y.

Pristker, A.A.B. (1986). Introduction to Simulation and SLAM-II. John and Sons, Inc., New York, N.Y.

Rueger, M., Behlau, T. (1995). "Create!: an Object-Oriented IDE for Discrete Event Simulation." Proceedings of the 1995 Winter Simulation Conference, Arlington, VA, USA.

Russel, A. Dubey, A. (1995). "Resource Leveling and Linear Scheduling." Proceedings of the Second Congress held in conjunction with A/E/C Systems '95 held in Atlanta, Georgia, June 5-8, 1995.

Sawhney, A., AbouRizk, S.M. (1996). "Computerized Tool for Hierarchical Simulation Modeling." Journal of Computing in Civil Engineering, 10(2), 115-124.

Shi, J., AbouRizk, S.M. (1997). "Resource-Based Modeling for Construction Simulation." Journal of Construction Engineering and Management, 123(1), 26-33.

Smith, M. R., and Collis, L. (1993). Aggregate:, Sand, Gravel And Crushed Rock Aggregates For Construction Purposes. The Geological Society, London, England.

Standridge, C. R. (1995). "Modular Modeling for Network Simulation Languages: Concepts and Examples." Proceedings of the 1995 Winter Simulation Conference, Arlington, VA, USA.

Takus, D.A., Profozich, D.M. (1997). "Arena Software Tutorial." Proceedings of the 1997 Winter Simulation Conference. Atlanta, GA, USA.

Thiem, G. (1906). Hydrologische Methoden, JM Gephardt, Leipzig.

Thomasma, T., Ulgen., O.,M. (1988). "Hierarchical, Modular Simulation Modeling in Icon-Based Simulation Program Generators for Manufacturing." Proceedings of the 1988 Winter Simulation Conference.

Tommelein, I.D., Carr, R.,I., Odeh, A.,M. (1994). "Knowledge-Based Assembly of Simulation Networks using Construction Designs, Plans, and Methods." Proceedings of the 1994 Winter Simulation Conference. Buena Vista, FL, USA.

Touran, A. (1989). "Expert System/Simulation Integration for Modeling Construction Operations." Computing in Civil Engineering. ASCE, 330-337. Sixth Conference on Computing in Civil Engineering, Atlanta, GA, USA.

Ulgen, O.M., Thomasma,T. (1986). "Simulation Modeling In An Object-Oriented Environment Using Smalltalk-80." Proceedings the 1986 Winter Simulation Conference Proceedings.

Ulgen, O.M., Thomasma,T, Mao,Y. (1989) "Object-Oriented Toolkits for Simulation Program Generators." Proceedings of the 1989 Winter Simulation Conference.

Ulgen, O.M., Thomasma, T., Otto, N. (1991). "Reusable Models: Making Your Models More User-Friendly." Proceedings of the 1991 Winter Simulation Conference.

Vaughn, W.R. (1998). Hitchhiker's Guide to Visual Basic and SQL Server, Sixth Edition, Microsoft Press, ISBN# 1-57231-848-1.

Wickard,D.,A, Bill,R.,D., Gates,K.H, Yoshinaga,T., Ohcoshi.,S. (1989) "Construction CAE. Integration of CAD, Simulation, Planning and Cost Control." Proceedings of the American Power Conference. V 51. Published by Illinois Inst of Technolgoy, Research Institute, Chicago, IL, USA. 983-987.

Zeigler, B.P. (1984). Multifaceted Modeling and Discrete Event Simulation. Academic Press, London and Orlando, Fla.

Zeigler, B.P. (1987). "Hierarchical, Modular Discrete-Event Modeling in an Object-Oriented Environment." Simulation, 49(5), 219-230.

Appendix 1 – Development Code for the Common Template

CreateEnt

```
Public Function CreateEnt_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single)
As Boolean

    ob.OnCreate x,y,True

    CreateEnt_OnCreate=True

    ob.AddAttribute "Quantity","Number of Entities to Create",CFC_Numeric, CFC_Single,
CFC_ReadWrite,1,1000
    ob.AddAttribute "First","Time of First Create",CFC_Numeric, CFC_Single,
CFC_ReadWrite,0,1000000
    ob.AddAttribute "Between","Time Between Creates",CFC_Distribution, CFC_Single,CFC_ReadWrite

    ob.AddAttribute "Fired","Entites CreateEntd so far",CFC_Numeric, CFC_Single,CFC_Hidden

    ob("Quantity")=1
    ob("First")=0
    ob("Between")=0

    ob.AddConnectionPoint "Out",x+57,y+25,COutput,5
End Function

Public Sub CreateEnt_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub CreateEnt_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.Circ ob.CoordinatesX(0)+25,ob.CoordinatesY(0)+25,25
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, ob("Quantity")

    ob.DrawConnectionPoints
End Sub

Public Sub CreateEnt_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "FireEntity"
End Sub

Public Sub CreateEnt_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance, RunNum As
Integer)
    ob.ScheduleEvent ob.AddEntity,"FireEntity",ob("First")
    ob("Fired")=0
End Sub

Public Sub CreateEnt_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent As
String, Entity As CFCSim_Entity)
    Dim newEntity As CFCSim_Entity
    If ob("Fired")>= ob("Quantity") Then Exit Sub
    ob("fired")=ob("fired")+1
    Set newEntity = ob.AddEntity
    ob.TransferOut NewEntity
    ob.ScheduleEvent entity, "FireEntity", ob("Between")

    Tracer.Trace "Entity: " & newEntity.Id & "  Created","Simulation"
End Sub
```

Destroy

```
Public Function Destroy_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    Dim cp As CFCSim_ConnectionPoint

    Destroy_OnCreate=True
    ob.OnCreate x,y,True
   ob.AddConnectionPoint "IN",x-7,y+25,CInput,5
End Function

Public Sub Destroy_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Destroy_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    ob.DeleteEntity entity
End Sub
```

SetAttribute

```
Option Explicit

Public Function SetAttribute_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    ob.OnCreate x,y,True
    Dim i As Integer
    SetAttribute_OnCreate=True
    For i=1 To 5 'up to 5 attributes
        ob.AddAttribute "Attr" & i & "Name","Attribute " & i &  " Name",CFC_Text, CFC_Single,
CFC_ReadWrite
        ob.AddAttribute "Attr" & i & "Val","Attribute "  & i &  " Value",CFC_Text, CFC_Single,
CFC_ReadWrite
        ob("Attr" & i & "Name")=""
    Next

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+100,y+25,COutput,5
End Function

Public Sub SetAttribute_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub
```

```
Public Sub SetAttribute_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",13,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+90,ob.CoordinatesY(0)+50
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+15, "   Set "
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+25, "Attributes "
    ob.DrawConnectionPoints
End Sub

Public Sub SetAttribute_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    Dim i As Integer
    For i=1 To 5
        If ob("Attr" & i & "Name")<>"" Then
            entity(ob("Attr" & i & "Name"))=ob("Attr" & i & "Val")
            Tracer.Trace "Entity: '" & entity.Id & "' has been assigned a value of '" & ob("Attr" & i &
"Val") & "' For attribute: '" & ob("Attr" & i & "Name") & "'", "Simulation"
        End If
    Next

    ob.TransferOut entity
End Sub
```

Branch

```
Public Function Branch_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    Dim cp As CFCSim_ConnectionPoint
    Branch_OnCreate=True
    With ob
         .OnCreate x,y,True

    .AddAttribute "Prob","Top Branch probability",CFC_Numeric,CFC_Single,CFC_ReadWrite,0,1
    .Attr("Prob")=0.5

    .AddConnectionPoint "IN",x-25,y,CInput,5
    .AddConnectionPoint "Top",x+25,y-25,COutput,5
    .AddConnectionPoint "Bottom",x+25,y+25,COutput,5
   End With
End Function

Public Sub Branch_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Branch_OnDraw(ob As CFCSim_ModelingElementInstance)


    CDC.MoveTo ob.ConnectionPoints("In").x,ob.ConnectionPoints("In").Y
    CDC.LineTo ob.CoordinatesX(0),ob.CoordinatesY(0)
    CDC.Arrow ob.CoordinatesX(0),ob.CoordinatesY(0)      ,
ob.ConnectionPoints("Top").x,ob.ConnectionPoints("Top").Y, 17
    CDC.Arrow ob.CoordinatesX(0), ob.CoordinatesY(0), ob.ConnectionPoints("Bottom").x ,
ob.ConnectionPoints("Bottom").Y, 17
```

```
    ob.DrawConnectionPoints
End Sub

Public Sub Branch_OnMove(ob As CFCSim_ModelingElementInstance, ByVal x1 As Single, ByVal y1 As
Single, ByVal x2 As Single, ByVal y2 As Single)


    Dim a As Double
    Dim b As Double
    Dim c As Double
    Dim X As Double
    Dim Ang As Double
    Dim xo As Double
    Dim yo As Double
    Dim cp As CFCSim_ConnectionPoint


    If Sqr((ob.CoordinatesX(0) - X1) ^ 2 + (ob.CoordinatesY(0) - Y1) ^ 2) > 20  Then
        a=  Sqr (  (ob.CoordinatesX(0)-x2) ^ 2 + (ob.CoordinatesY(0)-y2)^2 )
        b=  Sqr (  (ob.CoordinatesX(0)-x1) ^ 2 + (ob.CoordinatesY(0)-y1)^2 )
        c=  Sqr (  (x1-x2) ^ 2 + (y1-y2)^2 )

        'figure out  expression
        X= (c^2-a^2-b^2 ) / (-2*a*b)

        If Sqr(-X * X + 1)=0 Then Exit Sub
        'calculate inverse cosine
        Ang=Atn(-X / Sqr(-X * X + 1))

         ang=ang+ 2 * Atn(1)

        If ang=0 Then Exit Sub


        For Each cp In ob.ConnectionPoints
            xo=cp.x - ob.CoordinatesX(0)
            yo=- ( cp.Y - ob.CoordinatesY(0))
            cp.x = xo*Cos(ang)+yo*Sin(ang) +ob.CoordinatesX(0)
            cp.Y = -(-xo*Sin(ang)+yo*Cos(ang)) + ob.CoordinatesY(0)
        Next
    Else
        'call default on move
        ob.OnMove x1,y1,x2,y2, True
    End If

End Sub

Public Sub Branch_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    Dim sampled As Double
    sampled=Sampler.Uniform(0,1)


    If sampled< ob("Prob") Then
        ob.TransferOut entity, ob.ConnectionPoints("Top")
        Tracer.Trace "Routing to the top port" ,"Simulation"
    Else
        ob.TransferOut entity, ob.ConnectionPoints("Bottom")

        Tracer.Trace "Routing to the bottom port.","Simulation"
    End If
End Sub
```

Consolidate

```
Option Explicit

Public Function Consolidate_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    ob.OnCreate x,y,True
    Consolidate_OnCreate=True
    ob.AddAttribute "Quantity","Number to Consolidate",CFC_Numeric, CFC_Single,
CFC_ReadWrite,1,1000000
    ob.AddAttribute "Generate","Number of Entities to Generate",CFC_Numeric, CFC_Single,
CFC_ReadWrite,1,1000000

    ob.AddAttribute "CurrentLevel","",CFC_Numeric, CFC_Single, CFC_Hidden

    ob("Quantity")=5
    ob("Generate")=1

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out1",x+120,y+25,COutput,5
    ob.AddConnectionPoint "Out2",x+50,y+60,COutput,5
End Function

Public Sub Consolidate_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Consolidate_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",13,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+110,ob.CoordinatesY(0)+50
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+10, "Consolidate"
    If ob("Quantity").Calculation=CFC_Simple And ob("Generate").Calculation=CFC_Simple Then
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+25, " (" & ob("Quantity") & ") X " &
ob("Generate")
    Else
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+25, " ( Formula )"
    End If

    ob.DrawConnectionPoints
End Sub

Public Sub Consolidate_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance, RunNum As
Integer)
    ob("CurrentLevel")=0
```

```
End Sub

Public Sub Consolidate_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    Dim newEntity As CFCSim_Entity
    Dim i As Integer

    ob("CurrentLevel")=ob("CurrentLevel")+1
    If ob("CurrentLevel")=ob("Quantity") Then
        ob("CurrentLevel")=0

        For i=1 To Int(ob("Generate"))
            Set newEntity=ob.CloneEntity(entity)

            Tracer.Trace "New entity created: " & newEntity.Id,"Simulation"
            ob.TransferOut newentity, ob.ConnectionPoints("Out2")
        Next
    End If

    ob.TransferOut entity, ob.ConnectionPoints("Out1")
End Sub
```

Task

```
Public Function Task_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    ob.OnCreate x,y,True
    Task_OnCreate=True
    ob.AddAttribute "Duration","Delay Duration",CFC_Numeric, CFC_Single, CFC_ReadWrite,0
    ob("Duration")=1

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+115,y+25,COutput,5
End Function

Public Sub Task_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Task_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",13,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+100,ob.CoordinatesY(0)+50
    If ob("Duration").Calculation=CFC_Simple Then
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Task (" & ob("Duration") & ")"
    Else
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Task (Formula)"
    End If

    ob.DrawConnectionPoints
End Sub

Public Sub Task_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "Start",True
    ob.AddEvent "Finish"
End Sub

Public Sub Task_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent As
String, Entity As CFCSim_Entity)
    Dim Duration As Double

    Select Case MyEvent
    Case "Start"
        duration=ob("Duration")

        ob.ScheduleEvent entity,"Finish",duration

        Tracer.Trace "Entity: " & entity.ID & " will incurr a delay of " & duration,"Simulation"

    Case "Finish"
        Tracer.Trace "Delay of Entity: " & entity.ID & " Completed" , "Simulation"
        ob.TransferOut entity
    End Select
End Sub
```

RandomTask

```
Option Explicit

Public Function RandomTask_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    ob.OnCreate x,y,True
    RandomTask_OnCreate=True
    ob.AddAttribute "Duration","Delay Duration",CFC_Distribution, CFC_Single, CFC_ReadWrite

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+175,y+25,COutput,5
End Function

Public Sub RandomTask_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub RandomTask_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Arial",12,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+160,ob.CoordinatesY(0)+50

    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+10, "Task"
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+30,
ob("Duration").Distribution.GetStringRepresentation

    ob.DrawConnectionPoints
End Sub

Public Sub RandomTask_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "Start",True
    ob.AddEvent "Finish"
```

```
End Sub

Public Sub RandomTask_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent
As String, Entity As CFCSim_Entity)
    Dim Duration As Double

    Select Case MyEvent
    Case "Start"
        duration=ob("Duration")

        ob.ScheduleEvent entity,"Finish",duration

        Tracer.Trace "Entity: " & entity.ID & " will incurr a delay of " & duration,"Simulation"

    Case "Finish"
        Tracer.Trace "Delay of Entity: " & entity.ID & " Completed" , "Simulation"
        ob.TransferOut entity
    End Select
End Sub
```

Resource

```
Public Function Resource_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single)
As Boolean
    Resource_OnCreate=True
    ob.OnCreate x,y,True
    ob.AddAttribute "ResName","Resource Description",CFC_Text, CFC_Single, CFC_ReadWrite
    ob.AddAttribute "Total","Total Number of Resources",CFC_Numeric, CFC_Single,
CFC_ReadWrite,1,1000000
    ob.AddAttribute "Current","Current Number of Available Resources",CFC_Numeric, CFC_Single,
CFC_ReadOnly
    ob.AddAttribute "Requests","",CFC_Collection, CFC_Single, CFC_Hidden
    ob.AddAttribute "Winners","",CFC_Collection, CFC_Single, CFC_Hidden

    ob("ResName")= "Res"
    ob("Total")=1
    ob("Current")=1

    ob.AddStatistic "Utilization","Resource Utilization",True,False

End Function

Public Sub Resource_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",12,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+50,ob.CoordinatesY(0)+50
    CDC.TextOut ob.CoordinatesX(0)+3,ob.CoordinatesY(0)+15, ob("ResName")
    If ob("Total").Calculation=CFC_Simple Then
        CDC.TextOut ob.CoordinatesX(0)+10,ob.CoordinatesY(0)+25, " (" & ob("Total") & ")"
    Else
        CDC.TextOut ob.CoordinatesX(0)+10,ob.CoordinatesY(0)+25, "Formula"
    End If

    ob.DrawConnectionPoints
```

```
End Sub

Public Sub Resource_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "CollectStat"
    ob.AddEvent "Release"

    'find out which request nodes use us
    Dim request As CFCSim_ModelingElementInstance

    ClearCollection ob("Requests").Collection
    For Each request In ob.Parent.ChildElements
        If request.ElementType="Request" Then
            If request("ResOb").Reference Is ob Then
                ob("Requests").Collection.Add request
            End If
        End If
    Next
End Sub


Public Sub Resource_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance, RunNum As
Integer)
    ob("Current")=ob("total")
End Sub

Public Sub Resource_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent As
String, Entity As CFCSim_Entity)
    Select Case MyEvent
    Case "CollectStat"
        ob.stat("Utilization").Collect 100 * (1-(ob("Current")/ob("Total")))
        ob.DeleteEntity entity
    Case "Release"

        Dim request As CFCSim_ModelingElementInstance
        Dim WinnerRequest As CFCSim_ModelingElementInstance
        Dim WinnerPriority As Long
        ClearCollection ob("Winners").Collection

        ' First Find Out Highest Priority
        WinnerPriority=-1
        For Each request In ob("requests").Reference
            If request.File("ResFile").Length>0 Then
                request.File("ResFile").MoveFirst
                If WinnerPriority < request.File("resFile").Priority Then
                    WinnerPriority = request.File("resFile").Priority
                End If
            End If
        Next

        ' Then find out all the requests that have that priority
        For Each request In ob("requests").Reference
            If request.File("ResFile").Length>0 Then
                request.File("ResFile").MoveFirst
                If WinnerPriority = request.File("resFile").Priority Then
                    ob("Winners").Collection.Add request
                End If
            End If
        Next

        If ob("Winners").Collection.Count=0 Then
            ob.DeleteEntity entity
        ElseIf ob("Winners").Collection.Count=1 Then
            ob("Winners").Collection(1).ScheduleEvent entity,"Check",0
        Else
            ' resolve between requests randomly
            Dim x As Double
            x=Sampler.uniform(1,ob("Winners").Collection.Count+1)
            ob("Winners").Collection(Int(X)).ScheduleEvent entity,"Check",0
        End If

    End Select

End Sub
```

Capture

```
Public Function Capture_OnCheckIntegrity(ob As CFCSim_ModelingElementInstance) As Boolean
    Capture_OnCheckIntegrity=True

    'find resource
    Set ob("ResOB").Reference  = Nothing
    Dim childob As CFCSim_ModelingElementInstance
    For Each childob In ob.Parent.ChildElements
        If childob.ElementType="Resource" Then
            If childob("ResName")=ob("ResName") Then
                Set ob("ResOb").Reference=childob
            End If
        End If
    Next

    If ob("ResOB").Reference Is Nothing Then
        Tracer.Trace "Cannot find specified resource: " & ob("ResName"),"Integrity"
        Capture_OnCheckIntegrity=False
    End If
End Function

Public Function Capture_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    ob.OnCreate x,y,True
    Capture_OnCreate=True
    ob.AddAttribute "ResName","Resource Name",CFC_Text, CFC_ListBox, CFC_ReadWrite
    ob.AddAttribute "Quantity","Number of Resources To Capture",CFC_Numeric, CFC_Single,
CFC_ReadWrite,1
    ob.AddAttribute "Priority","Capture Priority",CFC_Numeric, CFC_Single, CFC_ReadWrite,1
    ob.AddAttribute "ResOb","",CFC_Object, CFC_Single, CFC_Hidden

    ob("Quantity")=1
    ob("Priority")=1

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+127,y+25,COutput,5
    ob.AddFile "ResFile",QUEUE
End Function

Public Sub Capture_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Capture_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",12,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+120,ob.CoordinatesY(0)+50

    If ob("Quantity").Calculation=CFC_Simple Then
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Capture:'" & ob("ResName") & "'
(" & ob("Quantity") & ")"
    Else
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Capture:'" & ob("ResName") & "'
(Formula)"
    End If


    ob.DrawConnectionPoints
End Sub

Public Sub Capture_OnListBoxInitialize(ob As CFCSim_ModelingElementInstance, attr As
CFCSim_Attribute, lstList As Object)
    Dim childob As CFCSim_ModelingElementInstance
    For Each childob In ob.Parent.ChildElements
        If childob.ElementType="Resource" Then
            lstlist.AddItem childob("ResName")
        End If
    Next
End Sub

Public Sub Capture_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "Check"
End Sub

Public Sub Capture_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent As
String, Entity As CFCSim_Entity)
    Dim res As CFCSim_ModelingElementInstance
    Set res=ob("ResOB").Reference
    Dim MainEntity As CFCSim_Entity

    ob.DeleteEntity entity

    If ob.File("ResFile").Length=0 Then Exit Sub
    ob.File("ResFile").MoveFirst
    If res("current") < ob.File("ResFile").entity("NumRes") Then
        Tracer.Trace "Not enough resoruces...","Simulation"
        Exit Sub
    End If

    ' release entity
    Set MainEntity= ob.File("resFile").Pop

    res("current")=res("current")- MainEntity("NumRes")

    ' tell resource to collect statistics
    'res.ScheduleEvent ob.AddEntity,"CollectStat",0

    Tracer.Trace "Entity " & MainEntity.Id & " entity obtained resource","Simulation"

    ob.TransferOut  MainEntity
End Sub

Public Sub Capture_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    entity("NumRes")=ob("Quantity")
    ob.File("Resfile").Add entity, ob("Priority")
    Ob.ScheduleEvent ob.AddEntity,"Check",0
End Sub
```

Release

```
Public Function Release_OnCheckIntegrity(ob As CFCSim_ModelingElementInstance) As Boolean
    Release_OnCheckIntegrity=True

    Set ob("ResOB").Reference= Nothing
    Dim childob As CFCSim_ModelingElementInstance
    For Each childob In ob.Parent.ChildElements
        If childob.ElementType="Resource" Then
            If childob("ResName")=ob("ResName") Then
                Set ob("ResOb").Reference=childob
            End If
        End If
    Next

    If ob("ResOB").Reference Is Nothing Then
        Tracer.Trace "Cannot find specified resource: " & ob("ResName"),"Integrity"
        Release_OnCheckIntegrity=False
    End If
End Function

Public Function Release_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    ob.OnCreate x,y,True
    Release_OnCreate=True
    ob.AddAttribute "ResName","Resource Name",CFC_Text, CFC_ListBox, CFC_ReadWrite
    ob.AddAttribute "Quantity","Number of Resources To Release",CFC_Numeric, CFC_Single,
CFC_ReadWrite,1,1000000
    ob.AddAttribute "ResOb","",CFC_Object, CFC_Single, CFC_Hidden

    ob("Quantity")=1

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+127,y+25,COutput,5
End Function
```

```
Public Sub Release_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Release_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",12,True,False,False,False
    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+120,ob.CoordinatesY(0)+50

    If ob("Quantity").Calculation=CFC_Simple Then
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Release:'" & ob("ResName") & "'("
& ob("Quantity") & ")"
    Else
        CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Release:'" & ob("ResName") & "'
(Formula)"
    End If

    ob.DrawConnectionPoints
End Sub

Public Sub Release_OnListBoxInitialize(ob As CFCSim_ModelingElementInstance, attr As
CFCSim_Attribute, lstList As Object)
    Dim childob As CFCSim_ModelingElementInstance
    For Each childob In ob.Parent.ChildElements
        If childob.ElementType="Resource" Then
            lstlist.AddItem childob("ResName")
        End If
    Next
End Sub

Public Sub Release_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "Release",True
End Sub

Public Sub Release_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance, MyEvent As
String, Entity As CFCSim_Entity)
    Dim res As CFCSim_ModelingElementInstance
    Dim Quantity As Integer
    Set res=ob("ResOb").Reference
    quantity=Int(ob("Quantity"))

    If (quantity+res("Current")) > res("Total") Then
        Tracer.Trace "Release will increase the number of available resources to " &
quantity+res("Current"), "Warning"
    End If

    res("Current")=res("Current") + quantity

    'send a release message to resource
    res.ScheduleEvent ob.AddEntity, "Release",0

    ob.TransferOut entity
End Sub
```

Statistic

```
Public Function Statistic_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean

    Statistic_OnCreate=True

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+100
    ob.CoordinatesY(1)=y+50

    ob.AddAttribute "StatName","Statistic's Name",CFC_Text, CFC_Single, CFC_ReadWrite
    ob.AddAttribute "Intrinsic","Intrinsic Statistic",CFC_Text, CFC_ListBox, CFC_ReadWrite
    ob.AddAttribute "Full","Do Full Tracking",CFC_Text, CFC_ListBox, CFC_ReadWrite

    ob("Intrinsic")="No"
    ob("Full")="Yes"

    ob("StatName") = ""
    ob.AddStatistic "Stat","Stat",False,True
    Statistic_OnCreate=True

End Function

Public Sub Statistic_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Statistic_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",12,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+100,ob.CoordinatesY(0)+50
    CDC.TextOut ob.CoordinatesX(0)+2,ob.CoordinatesY(0)+10, "Statistic"
    CDC.TextOut ob.CoordinatesX(0)+2,ob.CoordinatesY(0)+30, ob("StatName")

End Sub

Public Sub Statistic_OnListBoxInitialize(ob As CFCSim_ModelingElementInstance, attr As
CFCSim_Attribute, lstList As Object)
    lstlist.AddItem "Yes"
    lstlist.AddItem "No"
End Sub

Public Sub Statistic_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.stat("Stat").Intrinsic= (ob("Intrinsic")="Yes")
    ob.stat("Stat").TrackFull= (ob("Full")= "Yes")
End Sub

Public Function Statistic_OnValidateParameters(ob As CFCSim_ModelingElementInstance, Parameters
As Object) As Boolean
    Statistic_OnValidateParameters=False
    If Parameters("Intrinsic") = "Yes" And Parameters("Intrinsic") = "No" Then
        MessagePrompt "Please specirfy either 'Yes' or 'No' for the Intrinisc parameter"
        Exit Function
    End If

    If Parameters("Full") <> "Yes" And Parameters("Full") <> "No" Then
        MessagePrompt "Please specirfy either 'Yes' or 'No' for the Full Tracking parameter"
        Exit Function
    End If

    Statistic_OnValidateParameters=True

End Function
```

CollectStat

```
Public Function CollectStat_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single)
As Boolean
    ob.OnCreate x,y,True
    CollectStat_OnCreate=True
    ob.AddAttribute "StatName","Statistic",CFC_Text, CFC_ListBox, CFC_ReadWrite
    ob.AddAttribute "Value","Value to CollectStat",CFC_Numeric, CFC_Single, CFC_ReadWrite

    ob("value")=0

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+60,y+25,COutput,5
End Function

Public Sub CollectStat_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub CollectStat_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",12,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+50,ob.CoordinatesY(0)+50
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+10, "Collect"
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Value"
    ob.DrawConnectionPoints
End Sub

Public Sub CollectStat_OnListBoxInitialize(ob As CFCSim_ModelingElementInstance, attr As
CFCSim_Attribute, lstList As Object)
    Dim childob As CFCSim_ModelingElementInstance
    For Each childob In ob.Parent.ChildElements
        If childob.ElementType="Statistic" Then
            lstlist.AddItem childob("StatName")
        End If
    Next
End Sub

Public Sub CollectStat_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
```

```
    Dim value As Double
    Dim stat As CFCSim_Statistic

    'find statistic mentioned
    Set stat=Nothing

    Dim childob As CFCSim_ModelingElementInstance
    For Each childob In ob.Parent.ChildElements
        If childob.ElementType="Statistic" Then
            If childob("StatName")=ob("StatName") Then
                Set stat=childob.stat("stat")
                Exit For
            End If
        End If
    Next

    If stat Is Nothing Then
        Tracer.Trace "Cannot find specified statistic: " & ob("StatName") ,"Execution"
        Exit Sub
    End If

    value=ob("Value")
    stat.Collect value

    Tracer.Trace "Statistical value Collected: " & value, "Simulation"

    ob.TransferOut entity, ob.ConnectionPoints("Out")
End Sub
```

Connector

```
Public Function Connector_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single)
As Boolean
    ob.OnCreate x,y,True
    Connector_OnCreate=True

    ob.AddConnectionPoint "In" , x-3, y+3, CInput, 5
    ob.AddConnectionPoint "Out",x+3,y-3,COutput,5
End Function

Public Sub Connector_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Connector_OnDraw(ob As CFCSim_ModelingElementInstance)
    ob.DrawConnectionPoints
End Sub

Public Sub Connector_OnGetBoundingRect(ob As CFCSim_ModelingElementInstance, mRect As
CFCGraphics_Rect)
    mrect.left=ob.CoordinatesX(0)-10
    mrect.top=ob.CoordinatesY(0)-10
    mrect.right=ob.CoordinatesX(0)+10
    mrect.bottom=ob.CoordinatesY(0)+10
End Sub
```

```
Public Sub Connector_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    ob.TransferOut entity
End Sub
```

Trace

```
Public Function Trace_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    ob.OnCreate x,y,True
    Trace_OnCreate=True

    ob.AddAttribute "Expression","Expression to Trace", CFC_Text, CFC_Single,CFC_ReadWrite
    ob.AddAttribute "Category","Trace Category", CFC_Text, CFC_Single,CFC_ReadWrite

    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+60,y+25,COutput,5

End Function

Public Sub Trace_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Trace_OnDraw(ob As CFCSim_ModelingElementInstance)
    Dim mRect As New CFCGraphics_Rect
    With ob
        mRect.left = .CoordinatesX(0)
       mRect.top = .CoordinatesY(0)
       mRect.right = .CoordinatesX(0) + 50
       mRect.bottom = .CoordinatesY(0) + 50
    End With

   CDC.Rectangle mRect.left, mRect.top, mRect.right, mRect.bottom

   If ob.Selected Then
      CDC.Rectangle mRect.left + 1, mRect.top + 1, mRect.right - 1, mRect.bottom - 1
   End If

    CDC.TextOut ob.CoordinatesX(0)+3,ob.CoordinatesY(0)+10,"Trace"
    If ob("expression").Calculation=CFC_Simple Then
        CDC.TextOut ob.CoordinatesX(0)+3,ob.CoordinatesY(0)+25,ob("Expression")
    Else
        CDC.TextOut ob.CoordinatesX(0)+3,ob.CoordinatesY(0)+25,"Formula"
    End If

End Sub

Public Sub Trace_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    ob.TransferOut entity
    Tracer.Trace ob("Expression"),ob("Category")
End Sub
```

Execute

```
Public Function Execute_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    ob.OnCreate x,y,True

    Execute_OnCreate=True

    ob.AddAttribute "Expression","Expression To Execute",CFC_Text, CFC_Single, CFC_ReadWrite


    ob.AddConnectionPoint "In" , x-10, y+25, CInput, 5
    ob.AddConnectionPoint "Out",x+60,y+25,COutput,5
End Function

Public Sub Execute_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub Execute_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.ChangeFont "Courier New",12,True,False,False,False

    CDC.Rectangle
ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(0)+50,ob.CoordinatesY(0)+50
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+20, "Execute"
    ob.DrawConnectionPoints
End Sub

Public Sub Execute_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    Dim x
    ' reference the attribute so that any linked expression is evaluated
    x=ob("Expression")

    ob.TransferOut entity
End Sub
```

InPort

```
Public Function InPort_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    Dim y1 As Single
    Dim x1 As Single
    Dim mRect As New CFCGraphics_Rect
    Dim i As Integer

    Dim cp As CFCSim_ConnectionPoint
    InPort_OnCreate=True
    With ob
        .OnCreate x,y,True
    .AddConnectionPoint "IN",x-10,y+25,CInput,0
    .AddConnectionPoint "Out",x+60,y+25,COutput,5

    i=1
    ob.Parent.OnGetBoundingRect mRect
       x1=mRect.left-10
       y1=mRect.top-10

      For Each cp In ob.Parent.ConnectionPoints
         If cp.ctype=CInput Then
            i=i+1
            If cp.y>y1 Then
                y1=cp.y
                x1=cp.x
            End If
          End If
      Next

      y1=y1+15
      ob.Parent.AddConnectionPoint "In" & i, x1, y1, CInput, 5
      ob.Parent.AddRelation ob.Parent.ConnectionPoints("IN" & i), ob.ConnectionPoints("IN")

   End With
End Function

Public Sub InPort_OnDelete(ob As CFCSim_ModelingElementInstance)
       ob.Parent.DeleteConnectionPoint ob.ConnectionPoints("IN").RelationsFrom(1)
End Sub

Public Sub InPort_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub
```

OutPort

```
Public Function OutPort_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As
Boolean
    Dim y1 As Single
    Dim x1 As Single
    Dim mRect As New CFCGraphics_Rect
    Dim i As Integer
    OutPort_OnCreate=True
    Dim cp As CFCSim_ConnectionPoint

    With ob
        .OnCreate x,y,True
     .AddConnectionPoint "IN",x-7,y+25,CInput,5
    .AddConnectionPoint "Out",x+57,y+25,COutput,0

       ob.Parent.OnGetBoundingRect mRect
       x1=mRect.Right+10
       y1=mRect.Top-5
    i=1

      For Each cp In ob.Parent.ConnectionPoints
         If cp.ctype=COutput Then
            i=i+1
            If cp.Y>y1 Then
            y1=cp.Y
            x1=cp.x
          End If
        End If
      Next

      y1=y1+15
      ob.Parent.AddConnectionPoint "Out" & i, x1, y1, COutput, 5
      ob.AddRelation ob.ConnectionPoints("Out"), ob.Parent.ConnectionPoints("Out" & i)
    End With
End Function

Public Sub OutPort_OnDelete(ob As CFCSim_ModelingElementInstance)
    ob.Parent.DeleteConnectionPoint ob.ConnectionPoints("OUT").RelationsTo(1)
End Sub

Public Sub OutPort_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub
```

Appendix 2 – Simphony Build History

Build 2: (released March 5, 1999)
1) Fixed refresh problem
2) fixed problem with root element connection point crashing
(when InPort is added then delete at the root element)

Build 3: (released March 10, 1999)
1) Add LimitList, GraphType attributes
2) Improved the grid (Columns,...)
3) Replaced intrinsic collection with my own CFCSim_Collection class
4) Fixed some hdc refresh problems
5) Add CurrentRun statement
6) Add the ColumnLabel member property to the CFCSim_Attribute Class

Build 4: (released March 18, 1999)
1) changed list box allignment for CFC_ListBox type attributes. (Aligns to the right side of the grid now)
2) Quick-Fixed a problem with RenderPicture method which only appears on Win85/98 machines by truncating
the bottom 5% of any picture.
3) Added an image export functionality to the Bitmap DB editor.
4) Added the "number of events processed" so far as a simulation progress indicator.
5) OnCheckIntegrity is now called for all parents first, then the children. Also, it will now fail
when it encounteres the first False return value.
6) Fixed minor bug with column labels as used by graph based attributes.
7) Mark fixed freezing problem associated with priority based resource Requests

Build 5: (released April 2, 1999)

1) Fixed collection class problem (NewEnum enumarator was returning object instead of item)
2) Scheduling an event for a given entity will now delete any prior events foe that entity and give a warning
3) Deleting an entity will first delete any scheduled events for it.
4) Enabled Full menus for Sax control so now Search and replace features are accessible in the designer through the popup menu.
5) Added user-scripting capabilities to the Editor (Window->Script Window)
6) Maximized Designer Window on Startup.
7) Added Project Export and Import Capability to allow for the transfer of project data between databases.
8) Made the element form a topmost window.
9) Upgrade to Sax Engine Build 15.

Build 6: (released April 12, 1999)

1) Made Intrinsic and TrackFull Property of CFCSim_Statistic class public.
2) Fixed problem with saving and loading of distribution type attributes (Id's were not being generated).
3) Fixed problem with statistical analysis of intrinsic data. (changed temporary variables from long to double).
4) Set the Cancel button on the project info dialog box to default.

Build 7: (released April 15, 1999)
1) Modeling element library form is no longer a topmost window
2) fixed problem with OpenProject method of CFCSim_Conductor class

Build 8: (released May 10)
1) fixed problem with array types attribtues where a reference to the assigned value instead of its value was being stored.
2) fixed problem with entities not cloned if transferout is called with a specific connection point

Build 9: (released June 10)
1) fixed bug with overflow during parsing of modeling element code (for elements with code size > 32000 characters)
2) fixed trace reporting error.
previously, any tracing messages prior to simulation (during integrity checking) were being cleared.
3) Removed "ImmediateUpdate" trace features. (useless property) (also removed its reference from the manuals)
4) Optimized trace message generation code.
5) upgraded sax basic engine to version 5.2 (Build 3)

Build 10: (released June 15)
1) Trace messages which included the single quote character were producing an error - fixed.
2) Accessing object-based attributes after a project load caused an error - fixed.

Appendix 3 – CEM_EMS Template Code

CEM_EMS

```
Public Function CEM_EMS_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    CEM_EMS_OnCreate = True

    ob.SetNumCoordinates 2
    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+100
    ob.CoordinatesY(1)=y+50

    Dim ob1 As CFCSim_ModelingElementInstance
    Dim ob2 As CFCSim_ModelingElementInstance
    Dim ob3 As CFCSim_ModelingElementInstance
    Dim ob4 As CFCSim_ModelingElementInstance
    Dim ob5 As CFCSim_ModelingElementInstance

    Set ob1 = ob.AddElement("CEM_EMS_Source",71.0,169.0)
    ob1("AmountToHaul")=100000
    ob1.CoordinatesX(0)=71
    ob1.CoordinatesY(0)=169
    ob1.CoordinatesX(1)=181
    ob1.CoordinatesY(1)=209
    ob1.ConnectionPoints("In1").x=116
    ob1.ConnectionPoints("In1").Y=153
    ob1.ConnectionPoints("Out1").x=116
    ob1.ConnectionPoints("Out1").Y=229

    Set ob2 = ob.AddElement("CEM_EMS_Placement",586.0,186.0)
    ob2.CoordinatesX(0)=586
    ob2.CoordinatesY(0)=186
    ob2.CoordinatesX(1)=696
    ob2.CoordinatesY(1)=226
    ob2.ConnectionPoints("In1").x=631
    ob2.ConnectionPoints("In1").Y=246
    ob2.ConnectionPoints("Out1").x=631
    ob2.ConnectionPoints("Out1").Y=170

    Set ob3 = ob.AddElement("CEM_EMS_Truck",16.0,60.0)
    ob3("Type")="Caterpillar Model 777C"
    ob3("Quantity")=1
    ob3("Capacity")=45
    ob3("DumpingTime").Distribution.DistType=0
    ob3("DumpingTime").Distribution.Positive=True
    ob3("DumpingTime").Distribution.ParameterValue(0)=1
    ob3("IPriority")=1
    ob3("LPriority")=1
```

```
  ob3("Path")=1
  ob3.CoordinatesX(0)=16
  ob3.CoordinatesY(0)=60
  ob3.CoordinatesX(1)=96
  ob3.CoordinatesY(1)=110
  ob3.ConnectionPoints("c1").x=106
  ob3.ConnectionPoints("c1").Y=85

  Set ob4 = ob.AddElement("CEM_EMS_Road",117.0,230.0)
  ob4("Length")=1000
  ob4("Grade")=2
  ob4("RR")=2
  ob4.CoordinatesX(0)=117
  ob4.CoordinatesY(0)=230
  ob4.CoordinatesX(1)=631
  ob4.CoordinatesY(1)=249
  ob4.ConnectionPoints("c1").x=117
  ob4.ConnectionPoints("c1").Y=230
  ob4.ConnectionPoints("c2").x=631
  ob4.ConnectionPoints("c2").Y=249

  Set ob5 = ob.AddElement("CEM_EMS_Road",631.0,171.0)
  ob5("Length")=1000
  ob5("Grade")=2
  ob5("RR")=2
  ob5.CoordinatesX(0)=631
  ob5.CoordinatesY(0)=171
  ob5.CoordinatesX(1)=118
  ob5.CoordinatesY(1)=154
  ob5.ConnectionPoints("c1").x=631
  ob5.ConnectionPoints("c1").Y=171
  ob5.ConnectionPoints("c2").x=118
  ob5.ConnectionPoints("c2").Y=154


  ob1.AddRelation(ob1.ConnectionPoints("Out1"),ob4.ConnectionPoints("c1"))
  ob2.AddRelation(ob2.ConnectionPoints("Out1"),ob5.ConnectionPoints("c1"))
  ob3.AddRelation(ob3.ConnectionPoints("c1"),ob1.ConnectionPoints("In1"))
  ob4.AddRelation(ob4.ConnectionPoints("c2"),ob2.ConnectionPoints("In1"))
  ob5.AddRelation(ob5.ConnectionPoints("c2"),ob1.ConnectionPoints("In1"))

End Function

Public Sub CEM_EMS_OnDraw(ob As CFCSim_ModelingElementInstance)


    CDC.TextOut ob.CoordinatesX(0)+10,ob.CoordinatesY(0), "Earth-moving"
    CDC.TextOut ob.CoordinatesX(0)+15,ob.CoordinatesY(0)+20, "Simulation"

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID,1,RGB(255,0,0)
    End If
    CDC.Rectangle ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1),ob.CoordinatesY(1)
    ob.DrawConnectionPoints

End Sub
```

CEM_EMS_Source

```
Public Function CEM_EMS_Source_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As Boolean
    CEM_EMS_Source_OnCreate=True
    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+110
    ob.CoordinatesY(1)=y+40

    ob.AddAttribute "AmountToHaul", "Amount of Earth To Haul", CFC_Numeric, CFC_Single, CFC_ReadWrite,1000
    ob.AddAttribute "AmountLoaded", "Amount of Earth Loaded So Far", CFC_Numeric, CFC_Single, CFC_ReadOnly

    ob("AmountToHaul")=100000
   Dim ob1 As CFCSim_ModelingElementInstance
   Set ob1 = ob
   Dim ob2 As CFCSim_ModelingElementInstance
   Dim ob3 As CFCSim_ModelingElementInstance
   Dim ob4 As CFCSim_ModelingElementInstance
   Dim ob5 As CFCSim_ModelingElementInstance
   Dim ob6 As CFCSim_ModelingElementInstance
   Dim ob7 As CFCSim_ModelingElementInstance
   Dim ob8 As CFCSim_ModelingElementInstance

   Set ob2 = ob1.AddElement("CEM_EMS_Pile",281.0,101.0)
   ob2("StartingAmount")=0
   ob2.CoordinatesX(0)=281
   ob2.CoordinatesY(0)=101
   ob2.ConnectionPoints("AddIn").x=271
   ob2.ConnectionPoints("AddIn").Y=106
   ob2.ConnectionPoints("AddOut").x=246
   ob2.ConnectionPoints("AddOut").Y=146
   ob2.ConnectionPoints("GetIn").x=316
   ob2.ConnectionPoints("GetIn").Y=146
   ob2.ConnectionPoints("GetOut").x=291
   ob2.ConnectionPoints("GetOut").Y=106

   Set ob3 = ob1.AddElement("CEM_EMS_TrucksIn",244.5,258.6)
   ob3.CoordinatesX(0)=244.5454
   ob3.CoordinatesY(0)=258.6364
   ob3.ConnectionPoints("IN").x=234.5455
   ob3.ConnectionPoints("IN").Y=283.6364
   ob3.ConnectionPoints("Out").x=304.5455
   ob3.ConnectionPoints("Out").Y=283.6364

   Set ob4 = ob1.AddElement("CEM_EMS_TrucksOut",607.8,73.7)
   ob4.CoordinatesX(0)=607.8181
   ob4.CoordinatesY(0)=73.72725
   ob4.ConnectionPoints("IN").x=600.8182
   ob4.ConnectionPoints("IN").Y=98.72727
   ob4.ConnectionPoints("Out").x=664.8182
   ob4.ConnectionPoints("Out").Y=98.72727
```

```
Set ob5 = ob1.AddElement("CEM_EMS_Excavator",396.0,77.7)
ob5("Quantity")=1
ob5("Productivity").Distribution.DistType=0
ob5("Productivity").Distribution.Positive=True
ob5("Productivity").Distribution.ParameterValue(0)=1000
ob5.CoordinatesX(0)=395.9999
ob5.CoordinatesY(0)=77.72728
ob5.CoordinatesX(1)=475.9999
ob5.CoordinatesY(1)=127.7273
ob5.ConnectionPoints("c1").x=386
ob5.ConnectionPoints("c1").Y=102.7273
ob5.ConnectionPoints("c2").x=486
ob5.ConnectionPoints("c2").Y=102.7273

Set ob6 = ob1.AddElement("CEM_EMS_Preparation",149.0,43.0)
ob6("AmountToPrepare")=100000
ob6.CoordinatesX(0)=149
ob6.CoordinatesY(0)=43
ob6.CoordinatesX(1)=229
ob6.CoordinatesY(1)=83
ob6.ConnectionPoints("In1").x=139
ob6.ConnectionPoints("In1").Y=48
ob6.ConnectionPoints("Out1").x=239
ob6.ConnectionPoints("Out1").Y=53

Set ob7 = ob1.AddElement("CEM_EMS_Dozer",48.0,23.0)
ob7("Quantity")=1
ob7("Capacity")=5
ob7("Productivity").Distribution.DistType=0
ob7("Productivity").Distribution.Positive=True
ob7("Productivity").Distribution.ParameterValue(0)=1000
ob7.CoordinatesX(0)=48
ob7.CoordinatesY(0)=23
ob7.CoordinatesX(1)=128
ob7.CoordinatesY(1)=73
ob7.ConnectionPoints("c1").x=138
ob7.ConnectionPoints("c1").Y=48

Set ob8 = ob1.AddElement("Connector",133.0,139.0)
ob8.CoordinatesX(0)=133
ob8.CoordinatesY(0)=139
ob8.ConnectionPoints("In").x=130
ob8.ConnectionPoints("In").Y=142
ob8.ConnectionPoints("Out").x=136
ob8.ConnectionPoints("Out").Y=136


ob2.DeleteChildren
ob3.DeleteChildren
ob4.DeleteChildren
ob5.DeleteChildren
ob6.DeleteChildren
ob7.DeleteChildren
ob8.DeleteChildren
```

```
  Dim ob9 As CFCSim_ModelingElementInstance
  Dim ob10 As CFCSim_ModelingElementInstance
  Dim ob11 As CFCSim_ModelingElementInstance
  Dim ob12 As CFCSim_ModelingElementInstance
  Dim ob13 As CFCSim_ModelingElementInstance
  Dim ob14 As CFCSim_ModelingElementInstance
  Dim ob15 As CFCSim_ModelingElementInstance

  Set ob9 = ob6.AddElement("InPort",76.0,134.0)
  ob9.CoordinatesX(0)=76
  ob9.CoordinatesY(0)=134
  ob9.ConnectionPoints("IN").x=66
  ob9.ConnectionPoints("IN").Y=159
  ob9.ConnectionPoints("Out").x=136
  ob9.ConnectionPoints("Out").Y=159

  Set ob10 = ob6.AddElement("OutPort",717.0,195.0)
  ob10.CoordinatesX(0)=717
  ob10.CoordinatesY(0)=195
  ob10.ConnectionPoints("IN").x=710
  ob10.ConnectionPoints("IN").Y=220
  ob10.ConnectionPoints("Out").x=774
  ob10.ConnectionPoints("Out").Y=220

  Set ob11 = ob6.AddElement("Branch",163.0,161.0)
  ob11("Prob").Calculation=CFC_Formula
  ob11("Prob").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & "    If
ob.Parent(""AmountPrepared"") <  ob.Parent(""AmountToPrepare"") Then" & Chr$(13) & Chr$(10) & "
   FXXXX = 0" & Chr$(13) & Chr$(10) & "    Else" & Chr$(13) & Chr$(10) & "            FXXXX = 1" &
Chr$(13) & Chr$(10) & "    End If    " & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
  ob11.CoordinatesX(0)=163
  ob11.CoordinatesY(0)=161
  ob11.ConnectionPoints("IN").x=138.0218
  ob11.ConnectionPoints("IN").Y=159.957
  ob11.ConnectionPoints("Top").x=189.0213
  ob11.ConnectionPoints("Top").Y=137.0647
  ob11.ConnectionPoints("Bottom").x=186.9355
  ob11.ConnectionPoints("Bottom").Y=187.0212

  Set ob12 = ob6.AddElement("Destroy",600.0,29.0)
  ob12.CoordinatesX(0)=600
  ob12.CoordinatesY(0)=29
  ob12.ConnectionPoints("IN").x=593
  ob12.ConnectionPoints("IN").Y=54

  Set ob13 = ob6.AddElement("Execute",263.0,201.0)
  ob13("Expression").Calculation=CFC_Formula
  ob13("Expression").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) &
"   ob.Parent(""AmountPrepared"")=ob.Parent(""AmountPrepared"")+ob.CurrentEntity(""Capacity"")"
& Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
  ob13.CoordinatesX(0)=263
  ob13.CoordinatesY(0)=201
  ob13.ConnectionPoints("In").x=253
  ob13.ConnectionPoints("In").Y=226
  ob13.ConnectionPoints("Out").x=323
  ob13.ConnectionPoints("Out").Y=226
```

```
    Set ob14 = ob6.AddElement("Task",385.0,203.0)
    ob14("Duration").Calculation=CFC_Formula
    ob14("Duration").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & "
FXXXX = (ob.CurrentEntity(""Capacity"")/ ob.CurrentEntity(""Productivity"").Value) * 60" & Chr$(13)
& Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
    ob14.CoordinatesX(0)=385
    ob14.CoordinatesY(0)=203
    ob14.ConnectionPoints("In").x=375
    ob14.ConnectionPoints("In").Y=228
    ob14.ConnectionPoints("Out").x=500
    ob14.ConnectionPoints("Out").Y=228

    Set ob15 = ob6.AddElement("Execute",562.0,202.0)
    ob15("Expression").Calculation=CFC_Formula
    ob15("Expression").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) &
"   ob.CurrentEntity(""Payload"")=ob.CurrentEntity(""Capacity"")" & Chr$(13) & Chr$(10) & "    " &
Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
    ob15.CoordinatesX(0)=562
    ob15.CoordinatesY(0)=202
    ob15.ConnectionPoints("In").x=552
    ob15.ConnectionPoints("In").Y=227
    ob15.ConnectionPoints("Out").x=622
    ob15.ConnectionPoints("Out").Y=227


    ob9.DeleteChildren
    ob10.DeleteChildren
    ob11.DeleteChildren
    ob12.DeleteChildren
    ob13.DeleteChildren
    ob14.DeleteChildren
    ob15.DeleteChildren

    ob9.AddRelation(ob9.ConnectionPoints("Out"),ob11.ConnectionPoints("IN"))
    ob11.AddRelation(ob11.ConnectionPoints("Top"),ob12.ConnectionPoints("IN"))
    ob11.AddRelation(ob11.ConnectionPoints("Bottom"),ob13.ConnectionPoints("In"))
    ob13.AddRelation(ob13.ConnectionPoints("Out"),ob14.ConnectionPoints("In"))
    ob14.AddRelation(ob14.ConnectionPoints("Out"),ob15.ConnectionPoints("In"))
    ob15.AddRelation(ob15.ConnectionPoints("Out"),ob10.ConnectionPoints("IN"))

    ob2.AddRelation(ob2.ConnectionPoints("AddOut"),ob8.ConnectionPoints("In"))
    ob2.AddRelation(ob2.ConnectionPoints("GetOut"),ob5.ConnectionPoints("c1"))
    ob3.AddRelation(ob3.ConnectionPoints("Out"),ob2.ConnectionPoints("GetIn"))
    ob5.AddRelation(ob5.ConnectionPoints("c2"),ob4.ConnectionPoints("IN"))
    ob6.AddRelation(ob6.ConnectionPoints("Out1"),ob2.ConnectionPoints("AddIn"))
    ob7.AddRelation(ob7.ConnectionPoints("c1"),ob6.ConnectionPoints("In1"))
    ob8.AddRelation(ob8.ConnectionPoints("Out"),ob6.ConnectionPoints("In1"))

End Function

Public Sub CEM_EMS_Source_OnDraw(ob As CFCSim_ModelingElementInstance)
     Dim cp As CFCSim_ConnectionPoint

     CDC.RenderPicture "EMS_Source",ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1)-
ob.CoordinatesX(0),ob.CoordinatesY(1)-ob.CoordinatesY(0)
```

```
    If ob.Selected Then
        CDC.Rectangle ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1),ob.CoordinatesY(1)
    End If

    For Each cp In ob.ConnectionPoints
        If cp.ctype = COutput And cp.RelationsTo.Count>0 Then
            CDC.ChangeFillColor RGB(255,0,0)
        Else
            CDC.ChangeFillColor -1
        End If

        CDC.Circ cp.x,cp.Y,cp.tolerance
        If cp.ctype=COutput Then
            CDC.TextOut cp.x-3, cp.Y-19, cp.RelationsFrom(1).ModelingElement("Num")
        Else
            CDC.TextOut cp.x-3, cp.Y+5,cp.RelationsTo(1).ModelingElement("Num")
        End If
    Next

End Sub

Public Function CEM_EMS_Source_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Source_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Source_OnRelationValid=False
        Exit Function
    End If

    If dstcp.ModelingElement.ElementType="CEM_EMS_Source" Then
        MessagePrompt "Source element cannot be connected to other Source elements "
        CEM_EMS_Source_OnRelationValid=False
    End If
End Function

Public Sub CEM_EMS_Source_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance,
RunNum As Integer)
    ob("AmountLoaded")=0
End Sub
```

CEM_EMS_Placement

```
Public Function CEM_EMS_Placement_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single,
y As Single) As Boolean
    CEM_EMS_Placement_OnCreate=True
    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+110
    ob.CoordinatesY(1)=y+40
    ob.AddAttribute "AmountDumped", "Amount of Earth Dumped So Far", CFC_Numeric, CFC_Single,
CFC_ReadOnly

    Dim ob1 As CFCSim_ModelingElementInstance
    Set ob1 = ob
    Dim ob2 As CFCSim_ModelingElementInstance
    Dim ob3 As CFCSim_ModelingElementInstance
    Dim ob4 As CFCSim_ModelingElementInstance
    Dim ob5 As CFCSim_ModelingElementInstance
    Dim ob6 As CFCSim_ModelingElementInstance
    Dim ob7 As CFCSim_ModelingElementInstance

    Set ob2 = ob1.AddElement("CEM_EMS_Pile",376.0,94.0)
    ob2("StartingAmount")=0
    ob2.CoordinatesX(0)=376
    ob2.CoordinatesY(0)=94
    ob2.ConnectionPoints("AddIn").x=366
    ob2.ConnectionPoints("AddIn").Y=99
    ob2.ConnectionPoints("AddOut").x=341
    ob2.ConnectionPoints("AddOut").Y=139
    ob2.ConnectionPoints("GetIn").x=411
    ob2.ConnectionPoints("GetIn").Y=139
    ob2.ConnectionPoints("GetOut").x=386
    ob2.ConnectionPoints("GetOut").Y=99

    Set ob3 = ob1.AddElement("CEM_EMS_TrucksIn",51.0,24.0)
    ob3.CoordinatesX(0)=51
    ob3.CoordinatesY(0)=24
    ob3.ConnectionPoints("IN").x=41
    ob3.ConnectionPoints("IN").Y=49
    ob3.ConnectionPoints("Out").x=111
    ob3.ConnectionPoints("Out").Y=49

    Set ob4 = ob1.AddElement("CEM_EMS_TrucksOut",351.0,317.0)
    ob4.CoordinatesX(0)=351
    ob4.CoordinatesY(0)=317
    ob4.ConnectionPoints("IN").x=344
    ob4.ConnectionPoints("IN").Y=342
    ob4.ConnectionPoints("Out").x=408
    ob4.ConnectionPoints("Out").Y=342

    Set ob5 = ob1.AddElement("CEM_EMS_Dump",210.0,26.0)
    ob5("Quantity")=1
    ob5.CoordinatesX(0)=210
    ob5.CoordinatesY(0)=26
    ob5.CoordinatesX(1)=290
    ob5.CoordinatesY(1)=76
    ob5.ConnectionPoints("c1").x=200
    ob5.ConnectionPoints("c1").Y=51
    ob5.ConnectionPoints("c2").x=300
    ob5.ConnectionPoints("c2").Y=51

    Set ob6 = ob1.AddElement("CEM_EMS_Spreading",415.0,23.0)
    ob6.CoordinatesX(0)=415
    ob6.CoordinatesY(0)=23
```

```
  ob6.CoordinatesX(1)=495
  ob6.CoordinatesY(1)=63
  ob6.ConnectionPoints("In1").x=405
  ob6.ConnectionPoints("In1").Y=28
  ob6.ConnectionPoints("Out1").x=505
  ob6.ConnectionPoints("Out1").Y=33

  Set ob7 = ob1.AddElement("CEM_EMS_Dozer",377.0,193.0)
  ob7("Quantity")=1
  ob7("Capacity")=5
  ob7("Productivity").Distribution.DistType=0
  ob7("Productivity").Distribution.Positive=True
  ob7("Productivity").Distribution.ParameterValue(0)=500
  ob7.CoordinatesX(0)=377
  ob7.CoordinatesY(0)=193
  ob7.CoordinatesX(1)=457
  ob7.CoordinatesY(1)=243
  ob7.ConnectionPoints("c1").x=467
  ob7.ConnectionPoints("c1").Y=218


  ob2.DeleteChildren
  ob3.DeleteChildren
  ob4.DeleteChildren
  ob5.DeleteChildren
  ob6.DeleteChildren
  ob7.DeleteChildren

  Dim ob8 As CFCSim_ModelingElementInstance
  Dim ob9 As CFCSim_ModelingElementInstance
  Dim ob10 As CFCSim_ModelingElementInstance
  Dim ob11 As CFCSim_ModelingElementInstance

  Set ob8 = ob6.AddElement("InPort",62.0,82.0)
  ob8.CoordinatesX(0)=62
  ob8.CoordinatesY(0)=82
  ob8.ConnectionPoints("IN").x=52
  ob8.ConnectionPoints("IN").Y=107
  ob8.ConnectionPoints("Out").x=122
  ob8.ConnectionPoints("Out").Y=107

  Set ob9 = ob6.AddElement("OutPort",501.0,82.0)
  ob9.CoordinatesX(0)=501
  ob9.CoordinatesY(0)=82
  ob9.ConnectionPoints("IN").x=494
  ob9.ConnectionPoints("IN").Y=107
  ob9.ConnectionPoints("Out").x=558
  ob9.ConnectionPoints("Out").Y=107

  Set ob10 = ob6.AddElement("Task",197.0,85.0)
  ob10("Duration").Calculation=CFC_Formula
  ob10("Duration").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & "
FXXXX =  (ob.CurrentEntity(""Capacity"")/ ob.CurrentEntity(""Productivity"").Value) * 60" & Chr$(13)
& Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
  ob10.CoordinatesX(0)=197
  ob10.CoordinatesY(0)=85
  ob10.ConnectionPoints("In").x=187
  ob10.ConnectionPoints("In").Y=110
  ob10.ConnectionPoints("Out").x=312
  ob10.ConnectionPoints("Out").Y=110

  Set ob11 = ob6.AddElement("Execute",387.0,83.0)
  ob11("Expression").Calculation=CFC_Formula
  ob11("Expression").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) &
"    ob.Parent(""AmountSpread"")=ob.Parent(""AmountSpread"")+ob.CurrentEntity(""Payload"")" &
Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
  ob11.CoordinatesX(0)=387
  ob11.CoordinatesY(0)=83
  ob11.ConnectionPoints("In").x=377
  ob11.ConnectionPoints("In").Y=108
  ob11.ConnectionPoints("Out").x=447
  ob11.ConnectionPoints("Out").Y=108


  ob8.DeleteChildren
  ob9.DeleteChildren
  ob10.DeleteChildren
  ob11.DeleteChildren

  ob8.AddRelation(ob8.ConnectionPoints("Out"),ob10.ConnectionPoints("In"))
  ob10.AddRelation(ob10.ConnectionPoints("Out"),ob11.ConnectionPoints("In"))
  ob11.AddRelation(ob11.ConnectionPoints("Out"),ob9.ConnectionPoints("IN"))

  ob2.AddRelation(ob2.ConnectionPoints("AddOut"),ob4.ConnectionPoints("IN"))
  ob2.AddRelation(ob2.ConnectionPoints("GetOut"),ob6.ConnectionPoints("In1"))
  ob3.AddRelation(ob3.ConnectionPoints("Out"),ob5.ConnectionPoints("c1"))
  ob5.AddRelation(ob5.ConnectionPoints("c2"),ob2.ConnectionPoints("AddIn"))
  ob6.AddRelation(ob6.ConnectionPoints("Out1"),ob2.ConnectionPoints("GetIn"))
  ob7.AddRelation(ob7.ConnectionPoints("c1"),ob2.ConnectionPoints("GetIn"))

End Function

Public Sub CEM_EMS_Placement_OnDraw(ob As CFCSim_ModelingElementInstance)
    Dim cp As CFCSim_ConnectionPoint

    CDC.RenderPicture "EMS_Placement",ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1)-
ob.CoordinatesX(0),ob.CoordinatesY(1)-ob.CoordinatesY(0)

    If ob.Selected Then
        CDC.Rectangle ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1),ob.CoordinatesY(1)
    End If

    For Each cp In ob.ConnectionPoints
        If cp.ctype = COutput And cp.RelationsTo.Count>0 Then
            CDC.ChangeFillColor RGB(255,0,0)
        Else
            CDC.ChangeFillColor -1
        End If

        CDC.Circ cp.x,cp.Y,cp.tolerance
        If cp.ctype=CInput Then
            CDC.TextOut cp.x-3, cp.Y-19, cp.RelationsTo(1).ModelingElement("Num")
        Else
            CDC.TextOut cp.x-3, cp.Y+5,cp.RelationsFrom(1).ModelingElement("Num")
        End If
    Next

End Sub

Public Function CEM_EMS_Placement_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Placement_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Placement_OnRelationValid=False
        Exit Function
    End If

    If dstcp.ModelingElement.ElementType="CEM_Placement_Source" Then
        MessagePrompt "Placement element cannot be connected to other Placement elements "
        CEM_EMS_Placement_OnRelationValid=False
    End If
End Function

Public Sub CEM_EMS_Placement_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance,
RunNum As Integer)
    ob("AmountDumped")=0
End Sub
```

CEM_EMS_Truck

```
Public Function CEM_EMS_Truck_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    CEM_EMS_Truck_OnCreate=True

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+80
    ob.CoordinatesY(1)=y+50

    ob.AddAttribute "Type", "Truck Type", CFC_Text, CFC_ListBox, CFC_ReadWrite
    ob.AddAttribute "Quantity", "Number of Trucks", CFC_Numeric, CFC_Single, CFC_ReadWrite,1,100
    ob.AddAttribute "Capacity", "Truck Capacity in Cubic Metres", CFC_Numeric, CFC_Single,
CFC_ReadWrite,5,200
    ob.AddAttribute "DumpingTime", "Truck Dumping Duration", CFC_Distribution, CFC_Single,
CFC_ReadWrite
    ob.AddAttribute "IPriority", "Truck Priority at Intersections", CFC_Numeric, CFC_Single,
CFC_ReadWrite,1
    ob.AddAttribute "LPriority", "Truck Loading Priority ", CFC_Numeric, CFC_Single,
CFC_ReadWrite,1
    ob.AddAttribute "Path", "Path Number to Follow on Branches", CFC_Numeric, CFC_Single,
CFC_ReadWrite,1

    ob("Type")="Caterpillar Model 777C"
    ob("Type").LimitList=True

    ob("Quantity")=1
    ob("Capacity")=45
    ob("IPriority")=1
    ob("LPriority")=1
    ob("Path")=1

    With ob("DumpingTime").Distribution
        .DistType=CFC_Constant
        .ParameterValue(0)=1
    End With

    ob.AddConnectionPoint "c1", ob.CoordinatesX(0)+90,ob.CoordinatesY(0)+25,COutput,5

    ob.AddStatistic "CycleTime","Truck Cycle Time",False,True


End Function

Public Sub CEM_EMS_Truck_OnDraw(ob As CFCSim_ModelingElementInstance)

    CDC.RenderPicture "EMS_Truck",ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1)-
ob.CoordinatesX(0),ob.CoordinatesY(1)-ob.CoordinatesY(0)
    If ob.Selected Then
        CDC.Rectangle ob.CoordinatesX(0)-2,ob.CoordinatesY(0)-
2,ob.CoordinatesX(1)+2,ob.CoordinatesY(1)+2
    End If
    ob.DrawConnectionPoints
End Sub

Public Sub CEM_EMS_Truck_OnListBoxInitialize(ob As CFCSim_ModelingElementInstance, attr As
CFCSim_Attribute, lstList As Object)
    If attr.Name="Type" Then
        Dim myset As Recordset

        Set myset = SimphonyConnection.OpenRecordset("select * From EMS_TruckTypes",
dbOpenSnapshot)
        myset.MoveFirst
        While Not myset.EOF
         lstlist.AddItem myset!Description
         myset.MoveNext
        Wend
    End If
End Sub

Public Function CEM_EMS_Truck_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Truck_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Truck_OnRelationValid=False
    End If
End Function
```

```
Public Sub CEM_EMS_Truck_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance,
RunNum As Integer)
    Dim truck As CFCSim_Entity
    Dim i As Integer

    Dim myset As Recordset

   Set myset = SimphonyConnection.OpenRecordset("select * From EMS_TruckTypes where description
= '" & ob("Type") & "'", dbOpenSnapshot)

    For i=1 To ob("Quantity")
        Set truck=ob.AddEntity
        truck("capacity")=ob("capacity")
        Set truck("DumpingTime")=ob("DumpingTime").Distribution
        Set Truck("CycleStat")= ob.stat("CycleTime")
        truck("PayLoad")=0
        Truck("StartTime") = -1
        Truck("LPriority")=ob("LPriority")
        Truck("IPriority")=ob("IPriority")
        truck("Path")=ob("Path")

        'set speed parameters
        truck("MeanL")=myset!MeanL
        truck("StDevL")=myset!StDevL
        truck("ScaleFactorL")=myset!ScaleFactorL
        truck("YFactorL")=myset!YFactorL

        truck("MeanE")=myset!MeanE
        truck("StDevE")=myset!StDevE
        truck("ScaleFactorE")=myset!ScaleFactorE
        truck("YFactorE")=myset!YFactorE

        Tracer.Trace "Truck entity created and transferred: " & truck.ID , "Simulation"

        ob.TransferOut truck
    Next
End Sub
```

CEM_EMS_Road

```
Public Function CEM_EMS_Road_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    CEM_EMS_Road_OnCreate=True

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+50
    ob.CoordinatesY(1)=y+50

    ob.AddAttribute "Length", "Road Length in Metres", CFC_Numeric, CFC_Single, CFC_ReadWrite,5
    ob.AddAttribute "Grade", "Percent Grade Resistance", CFC_Numeric, CFC_Single, CFC_ReadWrite,-
30,30
```

```
    ob.AddAttribute "RR", "Percent Rolling Resistance", CFC_Numeric, CFC_Single,
CFC_ReadWrite,0,30

    ob.Attr("Length")=1000
    ob.Attr("Grade")=2
    ob.Attr("RR")=2

    ob.AddConnectionPoint "c1", ob.CoordinatesX(0),ob.CoordinatesY(0),CInput,5
    ob.AddConnectionPoint "c2", ob.CoordinatesX(1),ob.CoordinatesY(1),COutput,5

End Function

Public Sub CEM_EMS_Road_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub CEM_EMS_Road_OnDraw(ob As CFCSim_ModelingElementInstance)
    Dim ratio As Double
   Dim Length As Integer
   Dim xoffset As Integer
   Dim yoffset As Integer
   Dim x As Single
   Dim y As Single

    ob.DrawConnectionPoints

    With CDC
        Length = Sqr((ob.CoordinatesX(1) - ob.CoordinatesX(0)) ^ 2 + (ob.CoordinatesY(1) -
ob.CoordinatesY(0)) ^ 2)
    If length<5 Then Exit Sub
    ratio = Length / 5
        If ob.Selected Then
            CDC.ChangeLineStyle CFC_SOLID, 1,RGB(255,0,0)
        End If

        xoffset = (ob.CoordinatesY(1) - ob.CoordinatesY(0)) / ratio
        yoffset = (ob.CoordinatesX(1) - ob.CoordinatesX(0) )/ ratio


        .MoveTo ob.CoordinatesX(0) - xoffset, ob.CoordinatesY(0)+ yoffset
        .LineTo ob.CoordinatesX(1) - xoffset, ob.CoordinatesY(1) + yoffset

         .MoveTo ob.CoordinatesX(0) + xoffset, ob.CoordinatesY(0) - yoffset
        .LineTo  ob.CoordinatesX(1) + xoffset, ob.CoordinatesY(1) - yoffset

        .ArrowHead ob.CoordinatesX(0)  ,ob.CoordinatesY(0) , ob.CoordinatesX(1)  ,ob.CoordinatesY(1) ,20

    End With

End Sub

Public Function CEM_EMS_Road_OnHitTest(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    'we need to figure out the distance to the central line
    Dim slope As Double
    Dim Temp1 As Double
    Dim Temp2 As Double

    Dim mRect As New CFCGraphics_Rect
    Dim InRect As Boolean

    CEM_EMS_Road_OnHitTest=False

    'first check if point is inside the bounding rectangle
    ob.OnGetBoundingRect mRect
    If Not (x>mrect.left And x<mrect.right And y>mrect.top And y<mrect.bottom) Then
        Exit Function
    End If

    'Then check some special cases

    If (Abs((ob.CoordinatesX(1)-ob.CoordinatesX(0)))<0.1) Or (Abs((ob.CoordinatesY(1)-
ob.CoordinatesY(0)))<0.1) Then
        CEM_EMS_Road_OnHitTest=True
        Exit Function
    End If

    slope=(ob.CoordinatesY(1)-ob.CoordinatesY(0))/(ob.CoordinatesX(1)-ob.CoordinatesX(0))

     temp1 = (x/slope+y-ob.CoordinatesY(0)+slope*ob.CoordinatesX(0))/(slope+1/slope)
     temp2 = slope*(temp1-ob.CoordinatesX(0))+ob.CoordinatesY(0)

    If   ( Sqr( (temp1-x)^2)+  (temp2-y)^2) < 20 Then
        CEM_EMS_Road_OnHitTest=True
    End If

End Function

Public Sub CEM_EMS_Road_OnMove(ob As CFCSim_ModelingElementInstance, ByVal x1 As Single,
ByVal y1 As Single, ByVal x2 As Single, ByVal y2 As Single)
        With ob
            If Sqr((.CoordinatesX(0) - X1) ^ 2 + (.CoordinatesY(0) - Y1) ^ 2) <= 10 Then
              .CoordinatesX(0) = .CoordinatesX(0) + (X2 - X1)
              .CoordinatesY(0) = .CoordinatesY(0) + (Y2 - Y1)

               .ConnectionPoints("c1").x = .ConnectionPoints("c1").x + (X2 - X1)
              .ConnectionPoints("c1").Y = .ConnectionPoints("c1").Y + (Y2 - Y1)

            ElseIf Sqr((.CoordinatesX(1) - X1) ^ 2 + (.CoordinatesY(1) - Y1) ^ 2) <= 10 Then
              .CoordinatesX(1) = .CoordinatesX(1) + (X2 - X1)
              .CoordinatesY(1) = .CoordinatesY(1) + (Y2 - Y1)

                 .ConnectionPoints("c2").x = .ConnectionPoints("c2").x + (X2 - X1)
              .ConnectionPoints("c2").Y = .ConnectionPoints("c2").Y + (Y2 - Y1)

            Else
              .CoordinatesX(0) = .CoordinatesX(0) + (X2 - X1)
              .CoordinatesY(0) = .CoordinatesY(0) + (Y2 - Y1)
              .CoordinatesX(1) = .CoordinatesX(1) + (X2 - X1)
              .CoordinatesY(1) = .CoordinatesY(1) + (Y2 - Y1)

              .ConnectionPoints("c1").x = .ConnectionPoints("c1").x + (X2 - X1)
            .ConnectionPoints("c1").Y = .ConnectionPoints("c1").Y + (Y2 - Y1)
            .ConnectionPoints("c2").x = .ConnectionPoints("c2").x + (X2 - X1)
            .ConnectionPoints("c2").Y = .ConnectionPoints("c2").Y + (Y2 - Y1)
            End If
        End With

End Sub

Public Function CEM_EMS_Road_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Road_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Road_OnRelationValid=False
    End If

End Function

Public Sub CEM_EMS_Road_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "StartTravel",True
    ob.AddEvent "FinishTravel"
End Sub

Public Sub CEM_EMS_Road_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance,
MyEvent As String, Entity As CFCSim_Entity)
    Dim duration As Double
    Dim TotalResistance As Double
    Dim Speed As Double

    Select Case MyEvent
    Case "StartTravel"
        'calculate duration
        TotalResistance=Abs(ob("Grade")+ob("RR"))

        If entity("PayLoad") >0 Then
            Speed = entity("scalefactorL") * Exp(-((TotalResistance - entity("meanL")) ^ 2) /
entity("stdevL")) + entity("yfactorL")
        Else
            Speed = entity("scalefactorE") * Exp(-((TotalResistance - entity("meanE")) ^ 2) /
entity("stdevE")) + entity("yfactorE")
        End If

        duration = (ob("length")/ speed) * 60/1000

        ob.ScheduleEvent entity,"FinishTravel",duration

        Tracer.Trace "Truck: " & entity.ID & " will travel with a speed of " & speed & " for a duration of "
& duration,"Simulation"
    Case "FinishTravel"

        Tracer.Trace "Truck entity completed travel: " & entity.ID , "Simulation"

        ob.TransferOut entity
    End Select
End Sub
```

```
Public Function CEM_EMS_Road_OnValidateParameters(ob As CFCSim_ModelingElementInstance,
Parameters As Object) As Boolean
    CEM_EMS_Road_OnValidateParameters=ob.OnValidateParameters(Parameters,True)

    If CEM_EMS_Road_OnValidateParameters=False Then Exit Function

    If Abs(Parameters("Grade")+Parameters("RR"))>55 Then
        MessagePrompt "Total Resistance 'RR + Grade resistance' cannot exceed fiftyfive"
        CEM_EMS_Road_OnValidateParameters=False
    End If
End Function
```

CEM_EMS_Intersection

```
Public Function CEM_EMS_Intersection_OnCreate(ob As CFCSim_ModelingElementInstance, x As
Single, y As Single) As Boolean
    Dim cp As CFCSim_ConnectionPoint
    CEM_EMS_Intersection_OnCreate=True
    With ob
        .OnCreate x,y,True

        .AddAttribute "Delay","Expected Delay",CFC_Distribution,CFC_Single,CFC_ReadWrite
        With ob("Delay").Distribution
            .DistType=CFC_Normal
            .ParameterValue(0)=1
        End With

    .AddConnectionPoint "IN1",x,y-25,CInput,5
    .AddConnectionPoint "Out1",x,y+25,COutput,5
    .AddConnectionPoint "IN2",x-25,y,CInput,5
    .AddConnectionPoint "Out2",x+25,y,COutput,5

    .AddResource "Intersection",1
   End With
End Function

Public Sub CEM_EMS_Intersection_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub CEM_EMS_Intersection_OnDraw(ob As CFCSim_ModelingElementInstance)

    ob.DrawConnectionPoints

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID, 1,RGB(255,0,0)
    End If

    CDC.MoveTo ob.ConnectionPoints("In1").x, ob.ConnectionPoints("In1").Y
    CDC.LineTo ob.ConnectionPoints("Out1").x , ob.ConnectionPoints("Out1").Y

    CDC.MoveTo ob.ConnectionPoints("In2").x, ob.ConnectionPoints("In2").Y
    CDC.LineTo ob.ConnectionPoints("Out2").x , ob.ConnectionPoints("Out2").Y
```

?

```
    CDC.ArrowHead ob.CoordinatesX(0),ob.CoordinatesY(0), ob.ConnectionPoints("Out1").x ,
ob.ConnectionPoints("Out1").Y, 20

    CDC.ArrowHead ob.CoordinatesX(0),ob.CoordinatesY(0), ob.ConnectionPoints("Out2").x,
ob.ConnectionPoints("Out2").Y,20

    CDC.ChangeFillColor 0
    CDC.Circ  ob.CoordinatesX(0),ob.CoordinatesY(0),5

End Sub

Public Sub CEM_EMS_Intersection_OnMove(ob As CFCSim_ModelingElementInstance, ByVal x1 As
Single, ByVal y1 As Single, ByVal x2 As Single, ByVal y2 As Single)


    Dim a As Double
    Dim b As Double
    Dim c As Double
    Dim X As Double
    Dim Ang As Double
    Dim xo As Double
    Dim yo As Double
    Dim cp As CFCSim_ConnectionPoint


    If Sqr((ob.CoordinatesX(0) - X1) ^ 2 + (ob.CoordinatesY(0) - Y1) ^ 2) > 20  Then
        a=  Sqr (  (ob.CoordinatesX(0)-x2) ^ 2 + (ob.CoordinatesY(0)-y2)^2 )
        b=  Sqr (  (ob.CoordinatesX(0)-x1) ^ 2 + (ob.CoordinatesY(0)-y1)^2 )
        c=  Sqr (  (x1-x2) ^ 2 + (y1-y2)^2 )

        'figure out  expression
        X= (c^2-a^2-b^2 ) / (-2*a*b)

        If Sqr(-X * X + 1)=0 Then Exit Sub
        'calculate inverse cosine
        Ang=Atn(-X / Sqr(-X * X + 1))

         ang=ang+ 2 * Atn(1)

        If ang=0 Then Exit Sub

        ang=ang

        For Each cp In ob.ConnectionPoints
            xo=cp.x - ob.CoordinatesX(0)
            yo=- ( cp.Y - ob.CoordinatesY(0))
            cp.x = xo*Cos(ang)+yo*Sin(ang) +ob.CoordinatesX(0)
            cp.Y = -(-xo*Sin(ang)+yo*Cos(ang)) + ob.CoordinatesY(0)
        Next
    Else
        'call default on move
        ob.OnMove x1,y1,x2,y2, True
    End If

End Sub
```

```
Public Sub CEM_EMS_Intersection_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "Delay1",True
    ob.AddEvent "Request"
    ob.AddEvent "Release"
    ob.AddEvent "Delay2"
End Sub

Public Sub CEM_EMS_Intersection_OnSimulationProcessEvent(ob As
CFCSim_ModelingElementInstance, MyEvent As String, Entity As CFCSim_Entity)
    Dim duration As Double

    Select Case MyEvent
    Case "Delay1"
        ob.ScheduleEvent entity,"Request",ob("Delay")/2
    Case "Request"
        If ob.RequestResource("Intersection", entity, 1, entity("IPriority")) Then

            ob.ScheduleEvent entity,"Release",1/6.0

            Tracer.Trace "entity obtained intersection: " & entity.ID , "Simulation"
        Else
            Tracer.Trace "entity waiting for a intersection " & entity.ID , "Simulation"
        End If
    Case "Release"
        ob.ReleaseResource "Intersection", entity

        Tracer.Trace "entity Crossed Intersection: " & entity.ID , "Simulation"
        ob.ScheduleEvent entity,"Delay2",ob("Delay")/2

    Case "Delay2"
        If entity("InCP").Name="IN1" Then
            ob.TransferOut entity,ob.ConnectionPoints("Out1")
        Else
            ob.TransferOut entity,ob.ConnectionPoints("Out2")
        End If
    End Select
End Sub

Public Sub CEM_EMS_Intersection_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance,
Entity As CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    Set entity("InCp")=dstcp
    ob.OnSimulationTransferIn entity, srccp, dstcp,True
End Sub
```

CEM_EMS_Split

```
Public Function CEM_EMS_Split_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    Dim cp As CFCSim_ConnectionPoint
    CEM_EMS_Split_OnCreate=True
    With ob
        .OnCreate x,y,True
    .AddAttribute "Fork1","Path number for Fork 1",CFC_Numeric,CFC_Single,CFC_ReadWrite,1
    .AddAttribute "Fork2","Path number for Fork 2",CFC_Numeric,CFC_Single,CFC_ReadWrite,1

    ob("Fork1")=1
    ob("Fork2")=2

    .AddConnectionPoint "IN",x-25,y,CInput,5
    .AddConnectionPoint "Fork1",x+25,y-25,COutput,5
    .AddConnectionPoint "Fork2",x+25,y+25,COutput,5
  End With
End Function

Public Sub CEM_EMS_Split_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub CEM_EMS_Split_OnDraw(ob As CFCSim_ModelingElementInstance)
    Dim x As Single
    Dim y As Single

    ob.DrawConnectionPoints

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID, 1,RGB(255,0,0)
    End If

    CDC.MoveTo ob.ConnectionPoints("In").x,ob.ConnectionPoints("In").Y
    CDC.LineTo ob.CoordinatesX(0),ob.CoordinatesY(0)
    CDC.Arrow ob.CoordinatesX(0),ob.CoordinatesY(0) ,
ob.ConnectionPoints("Fork1").x,ob.ConnectionPoints("Fork1").Y, 17
    CDC.Arrow ob.CoordinatesX(0), ob.CoordinatesY(0), ob.ConnectionPoints("Fork2").x ,
ob.ConnectionPoints("Fork2").Y, 17

    CDC.ChangeTextColor RGB(255,0,0)
    If ob("Fork1").Calculation = CFC_Simple Then
        x= ob.CoordinatesX(0) +  (ob.ConnectionPoints("Fork1").x - ob.CoordinatesX(0) ) /2
        y= ob.CoordinatesY(0) +  (ob.ConnectionPoints("Fork1").Y - ob.CoordinatesY(0) ) /2
        CDC.TextOut x,y,ob("Fork1")
    End If

    If ob("Fork2").Calculation = CFC_Simple Then
        x= ob.CoordinatesX(0)+ (ob.ConnectionPoints("Fork2").x - ob.CoordinatesX(0) ) /2
        y=  ob.CoordinatesY(0)+ (ob.ConnectionPoints("Fork2").Y - ob.CoordinatesY(0) ) /2
        CDC.TextOut x,y,ob("Fork2")
    End If

End Sub

Public Sub CEM_EMS_Split_OnMove(ob As CFCSim_ModelingElementInstance, ByVal x1 As Single,
ByVal y1 As Single, ByVal x2 As Single, ByVal y2 As Single)

    Dim a As Double
    Dim b As Double
    Dim c As Double
    Dim X As Double
    Dim Ang As Double
    Dim xo As Double
    Dim yo As Double
    Dim cp As CFCSim_ConnectionPoint


    If Sqr((ob.CoordinatesX(0) - X1) ^ 2 + (ob.CoordinatesY(0) - Y1) ^ 2) > 20  Then
        a=  Sqr (  (ob.CoordinatesX(0)-x2) ^ 2 + (ob.CoordinatesY(0)-y2)^2 )
        b=  Sqr (  (ob.CoordinatesX(0)-x1) ^ 2 + (ob.CoordinatesY(0)-y1)^2 )
        c=  Sqr (  (x1-x2) ^ 2 + (y1-y2)^2 )

        'figure out  expression
        X= (c^2-a^2-b^2 ) / (-2*a*b)

        If Sqr(-X * X + 1)=0 Then Exit Sub
        ' calculate inverse cosine
        Ang=Atn(-X / Sqr(-X * X + 1))

         ang=ang+ 2 * Atn(1)

        If ang=0 Then Exit Sub


        For Each cp In ob.ConnectionPoints
            xo=cp.x - ob.CoordinatesX(0)
            yo=- ( cp.Y - ob.CoordinatesY(0))
            cp.x = xo*Cos(ang)+yo*Sin(ang) +ob.CoordinatesX(0)
            cp.Y = -(-xo*Sin(ang)+yo*Cos(ang)) + ob.CoordinatesY(0)
        Next
    Else
        ' call default on move
        ob.OnMove x1,y1,x2,y2, True
    End If

End Sub

Public Sub CEM_EMS_Split_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity
As CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)

    If entity("Path")= ob("Fork1") Then
        ob.TransferOut entity, ob.ConnectionPoints("Fork1")

        Tracer.Trace "Routing entity " &entity.ID & " to the fork 1" ,"Simulation"
    Else
        ob.TransferOut entity, ob.ConnectionPoints("Fork2")

        Tracer.Trace "Routing entity" &entity.ID & " to fork 2.","Simulation"
    End If
End Sub
```

CEM_EMS_Ext_Traffic

```
Public Function CEM_EMS_Ext_Traffic_OnCreate(ob As CFCSim_ModelingElementInstance, x As
Single, y As Single) As Boolean
```

```
    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+200
    ob.CoordinatesY(1)=y+40

    CEM_EMS_Ext_Traffic_OnCreate=True

    ob.AddAttribute "First","Time of First Arrival",CFC_Numeric, CFC_Single,
CFC_ReadWrite,0,1000000
    ob.AddAttribute "Between","Time Between Arrivals",CFC_Distribution, CFC_Single,CFC_ReadWrite
    ob.AddAttribute "IPriority", " Priority", CFC_Numeric, CFC_Single, CFC_ReadWrite,1

    ob("First")=0
    ob("IPriority")=1
    With ob("Between").Distribution
        .DistType=CFC_Exponential
        .ParameterValue(0)=5
    End With

    ob.AddConnectionPoint "Out",x+100,y+45,COutput,5
End Function

Public Sub CEM_EMS_Ext_Traffic_OnDraw(ob As CFCSim_ModelingElementInstance)

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID, 1,RGB(255,0,0)
    End If

    CDC.Ellipse ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1),ob.CoordinatesY(1)

    CDC.ChangeFont "Courier New",11, True, False, False, False
    CDC.TextOut ob.CoordinqtesX(0)+20,ob.CoordinatesY(0)+20,
ob("Between").Distribution.GetStringRepresentation

    ob.DrawConnectionPoints
End Sub

Public Function CEM_EMS_Ext_Traffic_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Ext_Traffic_OnRelationValid=True
    If dstcp.ModelingElement.ElementType<>"CEM_EMS_Intersection" Then
        MessagePrompt "This element can only be connected to an intersection"
        CEM_EMS_Ext_Traffic_OnRelationValid=False
    End If
End Function

Public Sub CEM_EMS_Ext_Traffic_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.AddEvent "FireEntity"
End Sub

Public Sub CEM_EMS_Ext_Traffic_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance,
RunNum As Integer)
    ob.ScheduleEvent ob.AddEntity,"FireEntity",ob("First")
End Sub
```

```
Public Sub CEM_EMS_Ext_Traffic_OnSimulationProcessEvent(ob As
CFCSim_ModelingElementInstance, MyEvent As String, Entity As CFCSim_Entity)
    Dim newEntity As CFCSim_Entity

    Set newEntity = ob.AddEntity
    newEntity("IPriority")=ob("IPriority")
    ob.TransferOut NewEntity

    ob.ScheduleEvent entity, "FireEntity", ob("Between")

    Tracer.Trace "Entity: " & newEntity.ID & "  Created","Simulation"
End Sub
```

CEM_EMS_Pile

```
Public Function CEM_EMS_Pile_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    ob.OnCreate x,y,True
    CEM_EMS_Pile_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Source" And ob.Parent.ElementType <>
"CEM_EMS_Placement" Then
        MessagePrompt "Pile element can only be defined as a child of a source or a placement element"

        CEM_EMS_Pile_OnCreate=False
        Exit Function
    End If

    ob.AddAttribute "StartingAmount","Starting Amount in Cubic
Metres",CFC_Numeric,CFC_Single,CFC_ReadWrite,0
    ob.AddAttribute "CurrentAmount","Current Amount in Cubic
Metres",CFC_Numeric,CFC_Single,CFC_ReadOnly,0

    ob("StartingAmount")=0
    ob.AddConnectionPoint "AddIn",x-10,y+5,CInput,5
    ob.AddConnectionPoint "AddOut",x-35,y+45,COutput,5

    ob.AddConnectionPoint "GetIn",x+35,y+45,CInput,5
    ob.AddConnectionPoint "GetOut",x+10,y+5,COutput,5

    ob.AddFile "Queue",QUEUE
End Function

Public Sub CEM_EMS_Pile_OnDraw(ob As CFCSim_ModelingElementInstance)
    Dim mRect As New CFCGraphics_Rect

    ob.OnGetBoundingRect mRect

   CDC.RenderPicture "EMS_Pile",ob.CoordinatesX(0)-25,ob.CoordinatesY(0),50,50

    CDC.ChangeFont "Courier New",12, True, False, False, False

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID, 1,RGB(255,0,0)
        CDC.Rectangle mRect.left,mrect.top,mrect.right,mrect.bottom
    End If

    If ob("StartingAmount").Calculation=CFC_Simple Then
        CDC.TextOut ob.CoordinatesX(0)-
16,ob.CoordinatesY(0)+42,Format(ob("StartingAmount"),"0.##")
    Else
        CDC.TextOut ob.CoordinatesX(0)-16,ob.CoordinatesY(0)+42,"Formula"
    End If
    ob.DrawConnectionPoints
End Sub

Public Sub CEM_EMS_Pile_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance,
RunNum As Integer)
    ob("CurrentAmount")=ob("StartingAmount")

End Sub

Public Sub CEM_EMS_Pile_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance, Entity As
CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
    Dim WaitingEntity As CFCSim_Entity

    If dstcp.Name="AddIn" Then
        ob("CurrentAmount")=ob("CurrentAmount")+ entity("Capacity")
        Tracer.Trace "Incoming Entity " & entity.ID & " Increased Amount by " & entity("Capacity")

        'check waiting entities
        With ob.File("Queue")
            If .Length>0 Then
                Do
                    .MoveFirst
                    If ob("CurrentAmount") > .entity("Capacity") Then
                    Set WaitingEntity = .Pop
                    ob("CurrentAmount") = ob("CurrentAmount") - WaitingEntity("Capacity")
                    ob.TransferOut WaitingEntity, ob.ConnectionPoints("GetOut")
                    Tracer.Trace "Queued Entity Freed " & WaitingEntity.ID
                    Else
                    Exit Do
                    End If
                Loop While .Length > 0
            End If
        End With
        ob.TransferOut entity, ob.ConnectionPoints("AddOut")
    Else
        If ob("CurrentAmount")< entity("Capacity") Then
            ob.File("Queue").Add entity
            Tracer.Trace "Incoming Entity Queued " & entity.ID
        Else
            ob("CurrentAmount") = ob("CurrentAmount") -  entity("Capacity")
            ob.TransferOut entity, ob.ConnectionPoints("GetOut")
            Tracer.Trace "Incoming Entity Served " & entity.ID
        End If
    End If
End Sub
```

CEM_EMS_TruckIn

```
Public Function CEM_EMS_TrucksIn_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y
As Single) As Boolean
    Dim y1 As Single
    Dim x1 As Single
    Dim mRect As New CFCGraphics_Rect
    Dim i As Integer
    Dim cp As CFCSim_ConnectionPoint

    CEM_EMS_TrucksIn_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Source" And ob.Parent.ElementType <>
"CEM_EMS_Placement" Then
        MessagePrompt "TrucksIn element can only be defined as a child of a source or a placement
element"

        CEM_EMS_TrucksIn_OnCreate=False
        Exit Function
    End If

    With ob
        .OnCreate x,y,True
    .AddAttribute "Num","",CFC_Numeric,CFC_Single,CFC_Hidden

    .AddConnectionPoint "IN",x-10,y+25,CInput,0
    .AddConnectionPoint "Out",x+60,y+25,COutput,5

    i=1
    ob.Parent.OnGetBoundingRect mRect
      x1=mRect.left+30
      If ob.Parent.ElementType="CEM_EMS_Source" Then
       y1=mRect.top-16
      Else
       y1=mRect.bottom+10
      End If

      For Each cp In ob.Parent.ConnectionPoints
        If cp.ctype=CInput Then
            i=i+1
            If cp.x>x1 Then
                x1=cp.x
                y1=cp.Y
            End If
          End If
      Next
      x1=x1+15
      ob.Parent.AddConnectionPoint "In" & i, x1, y1, CInput, 5
      ob.Parent.AddRelation ob.Parent.ConnectionPoints("IN" & i), ob.ConnectionPoints("IN")
      ob("Num")=i
   End With
End Function

Public Sub CEM_EMS_TrucksIn_OnDelete(ob As CFCSim_ModelingElementInstance)
   ob.Parent.DeleteConnectionPoint ob.ConnectionPoints("IN").RelationsFrom(1)
```

```
End Sub

Public Sub CEM_EMS_TrucksIn_OnDragDraw(ob As CFCSim_ModelingElementInstance)
    ob.OnDraw
End Sub

Public Sub CEM_EMS_TrucksIn_OnDraw(ob As CFCSim_ModelingElementInstance)
    ob.DrawConnectionPoints

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID,1, RGB(255,0,0)
    End If

    CDC.MoveTo ob.CoordinatesX(0), ob.CoordinatesY(0)
    CDC.LineTo  ob.CoordinatesX(0), ob.CoordinatesY(0)+50
    CDC.LineTo ob.CoordinatesX(0)+50, ob.CoordinatesY(0)+25
    CDC.LineTo ob.CoordinatesX(0), ob.CoordinatesY(0)

    CDC.TextOut ob.CoordinatesX(0)+20,ob.CoordinatesY(0)+20,CStr(ob("Num"))
End Sub
```

CEM_EMS_TruckOut

```
Public Function CEM_EMS_TrucksOut_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single,
y As Single) As Boolean
    Dim y1 As Single
    Dim x1 As Single
    Dim mRect As New CFCGraphics_Rect
    Dim i As Integer
    Dim cp As CFCSim_ConnectionPoint

    CEM_EMS_TrucksOut_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Source" And ob.Parent.ElementType <>
"CEM_EMS_Placement" Then
        MessagePrompt "TrucksOut element can only be defined as a child of a source or a placement
element"

        CEM_EMS_TrucksOut_OnCreate=False
        Exit Function
    End If

    With ob
        .OnCreate x,y,True
      .AddConnectionPoint "IN",x-7,y+25,CInput,5
    .AddConnectionPoint "Out",x+57,y+25,COutput,0

      .AddAttribute "Num","",CFC_Numeric,CFC_Single,CFC_Hidden

    i=1
    ob.Parent.OnGetBoundingRect mRect
      x1=mRect.left+30

       If ob.Parent.ElementType="CEM_EMS_Placement" Then
      y1=mRect.top-16
    Else
      y1=mRect.bottom+10
    End If


    For Each cp In ob.Parent.ConnectionPoints
       If cp.ctype=COutput Then
          i=i+1
          If cp.x>x1 Then
               x1=cp.x
               y1=cp.Y
          End If
       End If
    Next
      x1=x1+15

    ob.Parent.AddConnectionPoint "Out" & i, x1, y1, COutput, 5
    ob.AddRelation ob.ConnectionPoints("Out"), ob.Parent.ConnectionPoints("Out" & i)
    ob("Num")=i
  End With


End Function

Public Sub CEM_EMS_TrucksOut_OnDelete(ob As CFCSim_ModelingElementInstance)
       ob.Parent.DeleteConnectionPoint ob.ConnectionPoints("OUT").RelationsTo(1)
End Sub

Public Sub CEM_EMS_TrucksOut_OnDraw(ob As CFCSim_ModelingElementInstance)
    ob.DrawConnectionPoints

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID,1, RGB(255,0,0)
    End If

    CDC.MoveTo ob.CoordinatesX(0)+50 ,ob.CoordinatesY(0)
    CDC.LineTo  ob.CoordinatesX(0)+50, ob.CoordinatesY(0)+50
    CDC.LineTo ob.CoordinatesX(0), ob.CoordinatesY(0)+25
    CDC.LineTo ob.CoordinatesX(0)+50, ob.CoordinatesY(0)


    CDC.TextOut ob.CoordinatesX(0)+20,ob.CoordinatesY(0)+20,CStr(ob("Num"))

End Sub
```

CEM_EMS_Dozer

```
Public Function CEM_EMS_Dozer_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
    CEM_EMS_Dozer_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Source" And ob.Parent.ElementType <>
"CEM_EMS_Placement" Then
```

```
        MessagePrompt "Dozers can only be defined as a child of a source or a placement element"

        CEM_EMS_Dozer_OnCreate=False
        Exit Function
    End If

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+80
    ob.CoordinatesY(1)=y+50

    ob.AddAttribute "Quantity", "Number of Dozers", CFC_Numeric, CFC_Single, CFC_ReadWrite,1,100
    ob.AddAttribute "Capacity", "Capacity in Cubic Metres", CFC_Numeric, CFC_Single,
CFC_ReadWrite,0.1,200
    ob.AddAttribute "Productivity", "Productivity in Cubic Metres per Hr", CFC_Distribution,
CFC_Single, CFC_ReadWrite

    ob("Quantity")=1
    ob("Capacity")=5

    With ob("Productivity").Distribution
        .DistType=CFC_Constant
        .ParameterValue(0)=500
    End With

    ob.AddConnectionPoint "c1", ob.CoordinatesX(0)+90,ob.CoordinatesY(0)+25,COutput,5
End Function

Public Sub CEM_EMS_Dozer_OnDraw(ob As CFCSim_ModelingElementInstance)

    CDC.RenderPicture "EMS_Dozer",ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1)-
ob.CoordinatesX(0),ob.CoordinatesY(1)-ob.CoordinatesY(0)
    If ob.Selected Then
        CDC.Rectangle ob.CoordinatesX(0)-2,ob.CoordinatesY(0)-
2,ob.CoordinatesX(1)+2,ob.CoordinatesY(1)+2
    End If
    ob.DrawConnectionPoints
End Sub

Public Function CEM_EMS_Dozer_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Dozer_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Dozer_OnRelationValid=False
        Exit Function
    End If

    If dstCP.ModelingElement.ElementType<>"CEM_EMS_Preparation" And
dstCP.ModelingElement.ElementType<>"CEM_EMS_Pile" Then
        MessagePrompt "Dozers can only be connected to preparation or Pile elements"
        CEM_EMS_Dozer_OnRelationValid=False
        Exit Function
    End If
    If dstCP.ModelingElement.ElementType = "CEM_EMS_Pile" And dstcp.Name <> "GetIn" Then
        MessagePrompt "Dozers must be added to the input connection on the side of the pile"
        CEM_EMS_Dozer_OnRelationValid=False
        Exit Function
    End If
End Function

Public Sub CEM_EMS_Dozer_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance,
RunNum As Integer)
    Dim dozer As CFCSim_Entity
    Dim i As Integer


    For i=1 To ob("Quantity")
        Set dozer=          ob.AddEntity
        dozer("capacity")=ob("capacity")
        Set dozer("Productivity")=ob("Productivity").Distribution
        dozer("PayLoad")=0

        Tracer.Trace "dozer entity created and transferred: " & dozer.ID , "Simulation"
        ob.TransferOut dozer
    Next
End Sub
```

CEM_EMS_Preparation

```
Public Function CEM_EMS_Preparation_OnCreate(ob As CFCSim_ModelingElementInstance, x As
Single, y As Single) As Boolean
    CEM_EMS_Preparation_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Source" Then
        MessagePrompt "Preparation element can only be defined as a child of a source element"

        CEM_EMS_Preparation_OnCreate=False
        Exit Function
    End If

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+80
    ob.CoordinatesY(1)=y+40

    ob.AddAttribute "AmountToPrepare", "Amount of Earth To Prepare", CFC_Numeric, CFC_Single,
CFC_ReadWrite,1000
    ob.AddAttribute "AmountPrepared", "Amount of Earth Prepared so Far", CFC_Numeric, CFC_Single,
CFC_ReadOnly

    ob("AmountToPrepare")=100000

    Dim ob1 As CFCSim_ModelingElementInstance
   Set ob1 = ob
   Dim ob2 As CFCSim_ModelingElementInstance
   Dim ob3 As CFCSim_ModelingElementInstance
   Dim ob4 As CFCSim_ModelingElementInstance
   Dim ob5 As CFCSim_ModelingElementInstance
   Dim ob6 As CFCSim_ModelingElementInstance
   Dim ob7 As CFCSim_ModelingElementInstance
   Dim ob8 As CFCSim_ModelingElementInstance

   Set ob2 = ob1.AddElement("InPort",76.0,134.0)
   ob2.CoordinatesX(0)=76
   ob2.CoordinatesY(0)=134
   ob2.ConnectionPoints("IN").x=66
   ob2.ConnectionPoints("IN").Y=159
   ob2.ConnectionPoints("Out").x=136
   ob2.ConnectionPoints("Out").Y=159

   Set ob3 = ob1.AddElement("OutPort",717.0,195.0)
   ob3.CoordinatesX(0)=717
   ob3.CoordinatesY(0)=195
   ob3.ConnectionPoints("IN").x=710
   ob3.ConnectionPoints("IN").Y=220
   ob3.ConnectionPoints("Out").x=774
   ob3.ConnectionPoints("Out").Y=220

   Set ob4 = ob1.AddElement("Branch",163.0,161.0)
   ob4("Prob").Calculation=CFC_Formula
   ob4("Prob").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & "" &
Chr$(13) & Chr$(10) & "    If ob.Parent(""AmountPrepared"") <  ob.Parent(""AmountToPrepare"")
Then" & Chr$(13) & Chr$(10) & "              FXXXX = 0" & Chr$(13) & Chr$(10) & "    Else" & Chr$(13)
& Chr$(10) & "              FXXXX = 1" & Chr$(13) & Chr$(10) & "    End If    " & Chr$(13) & Chr$(10)
& "" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
   ob4.CoordinatesX(0)=163
   ob4.CoordinatesY(0)=161
   ob4.ConnectionPoints("IN").x=138.0218
   ob4.ConnectionPoints("IN").Y=159.957
   ob4.ConnectionPoints("Top").x=189.0213
   ob4.ConnectionPoints("Top").Y=137.0647
   ob4.ConnectionPoints("Bottom").x=186.9355
   ob4.ConnectionPoints("Bottom").Y=187.0212

   Set ob5 = ob1.AddElement("Destroy",600.0,29.0)
   ob5.CoordinatesX(0)=600
   ob5.CoordinatesY(0)=29
   ob5.ConnectionPoints("IN").x=593
   ob5.ConnectionPoints("IN").Y=54

   Set ob6 = ob1.AddElement("Execute",263.0,201.0)
   ob6("Expression").Calculation=CFC_Formula
   ob6("Expression").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
& Chr$(13) & Chr$(10) & "
ob.Parent(""AmountPrepared"")=ob.Parent(""AmountPrepared"")+ob.CurrentEntity(""Capacity"")" &
Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
   ob6.CoordinatesX(0)=263
   ob6.CoordinatesY(0)=201
   ob6.ConnectionPoints("In").x=253
   ob6.ConnectionPoints("In").Y=226
   ob6.ConnectionPoints("Out").x=323
   ob6.ConnectionPoints("Out").Y=226
```

```
  Set ob7 = ob1.AddElement("Task",385.0,203.0)
  ob7("Duration").Calculation=CFC_Formula
  ob7("Duration").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
& Chr$(13) & Chr$(10) & "    FXXXX = (ob.CurrentEntity(""Capacity"")/
ob.CurrentEntity(""Productivity"").Value) * 60" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) &
"" & Chr$(13) & Chr$(10) & ""
  ob7.CoordinatesX(0)=385
  ob7.CoordinatesY(0)=203
  ob7.ConnectionPoints("In").x=375
  ob7.ConnectionPoints("In").Y=228
  ob7.ConnectionPoints("Out").x=500
  ob7.ConnectionPoints("Out").Y=228

  Set ob8 = ob1.AddElement("Execute",562.0,202.0)
  ob8("Expression").Calculation=CFC_Formula
  ob8("Expression").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
& Chr$(13) & Chr$(10) & "    ob.CurrentEntity(""Payload"")=ob.CurrentEntity(""Capacity"")" &
Chr$(13) & Chr$(10) & "    " & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & "" & Chr$(13) &
Chr$(10) & ""
  ob8.CoordinatesX(0)=562
  ob8.CoordinatesY(0)=202
  ob8.ConnectionPoints("In").x=552
  ob8.ConnectionPoints("In").Y=227
  ob8.ConnectionPoints("Out").x=622
  ob8.ConnectionPoints("Out").Y=227


  ob2.DeleteChildren
  ob3.DeleteChildren
  ob4.DeleteChildren
  ob5.DeleteChildren
  ob6.DeleteChildren
  ob7.DeleteChildren
  ob8.DeleteChildren

  ob2.AddRelation(ob2.ConnectionPoints("Out"),ob4.ConnectionPoints("IN"))
  ob4.AddRelation(ob4.ConnectionPoints("Top"),ob5.ConnectionPoints("IN"))
  ob4.AddRelation(ob4.ConnectionPoints("Bottom"),ob6.ConnectionPoints("In"))
  ob6.AddRelation(ob6.ConnectionPoints("Out"),ob7.ConnectionPoints("In"))
  ob7.AddRelation(ob7.ConnectionPoints("Out"),ob8.ConnectionPoints("In"))
  ob8.AddRelation(ob8.ConnectionPoints("Out"),ob3.ConnectionPoints("IN"))


End Function

Public Sub CEM_EMS_Preparation_OnDraw(ob As CFCSim_ModelingElementInstance)

   CDC.ChangeFont "Courier New",13, True, False, False, False

   If ob.Selected Then
      CDC.ChangeLineStyle CFC_SOLID,1,RGB(255,0,0)
   End If

   CDC.Rectangle ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1),ob.CoordinatesY(1)
```

```
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+15, "Preparation"

    ob.DrawConnectionPoints

End Sub

Public Function CEM_EMS_Preparation_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Preparation_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Preparation_OnRelationValid=False
        Exit Function
    End If

End Function

Public Sub CEM_EMS_Preparation_OnSimulationInitializeRun(ob As
CFCSim_ModelingElementInstance, RunNum As Integer)
    ob("AmountPrepared")=0
End Sub
```

CEM_EMS_Spreading

```
Public Function CEM_EMS_Spreading_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single,
y As Single) As Boolean
    CEM_EMS_Spreading_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Placement" Then
        MessagePrompt "Spreading element can only be defined as a child of a placement element"

        CEM_EMS_Spreading_OnCreate=False
        Exit Function
    End If
    ob.AddAttribute "AmountSpread", "Amount of Earth Spread so Far", CFC_Numeric, CFC_Single,
CFC_ReadOnly

    Dim ob1 As CFCSim_ModelingElementInstance
   Set ob1 = ob
   Dim ob2 As CFCSim_ModelingElementInstance
   Dim ob3 As CFCSim_ModelingElementInstance
   Dim ob4 As CFCSim_ModelingElementInstance
   Dim ob5 As CFCSim_ModelingElementInstance

   Set ob2 = ob1.AddElement("InPort",62.0,82.0)
   ob2.CoordinatesX(0)=62
   ob2.CoordinatesY(0)=82
   ob2.ConnectionPoints("IN").x=52
   ob2.ConnectionPoints("IN").Y=107
   ob2.ConnectionPoints("Out").x=122
   ob2.ConnectionPoints("Out").Y=107

   Set ob3 = ob1.AddElement("OutPort",501.0,82.0)
   ob3.CoordinatesX(0)=501
   ob3.CoordinatesY(0)=82
   ob3.ConnectionPoints("IN").x=494
   ob3.ConnectionPoints("IN").Y=107
   ob3.ConnectionPoints("Out").x=558
   ob3.ConnectionPoints("Out").Y=107

   Set ob4 = ob1.AddElement("Task",197.0,85.0)
   ob4("Duration").Calculation=CFC_Formula
   ob4("Duration").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
& Chr$(13) & Chr$(10) & "    FXXXX =  (ob.CurrentEntity(""Capacity"")/
ob.CurrentEntity(""Productivity"").Value) * 60" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) &
"" & Chr$(13) & Chr$(10) & ""
   ob4.CoordinatesX(0)=197
   ob4.CoordinatesY(0)=85
   ob4.ConnectionPoints("In").x=187
   ob4.ConnectionPoints("In").Y=110
   ob4.ConnectionPoints("Out").x=312
   ob4.ConnectionPoints("Out").Y=110

   Set ob5 = ob1.AddElement("Execute",387.0,83.0)
   ob5("Expression").Calculation=CFC_Formula
   ob5("Expression").Formula.SetExpression"" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
& Chr$(13) & Chr$(10) & "
ob.Parent(""AmountSpread"")=ob.Parent(""AmountSpread"")+ob.CurrentEntity(""Payload"")" &
Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & "" & Chr$(13) & Chr$(10) & ""
   ob5.CoordinatesX(0)=387
   ob5.CoordinatesY(0)=83
   ob5.ConnectionPoints("In").x=377
   ob5.ConnectionPoints("In").Y=108
   ob5.ConnectionPoints("Out").x=447
   ob5.ConnectionPoints("Out").Y=108


   ob2.DeleteChildren
   ob3.DeleteChildren
   ob4.DeleteChildren
   ob5.DeleteChildren

   ob2.AddRelation(ob2.ConnectionPoints("Out"),ob4.ConnectionPoints("In"))
   ob4.AddRelation(ob4.ConnectionPoints("Out"),ob5.ConnectionPoints("In"))
   ob5.AddRelation(ob5.ConnectionPoints("Out"),ob3.ConnectionPoints("IN"))


End Function

Public Sub CEM_EMS_Spreading_OnDraw(ob As CFCSim_ModelingElementInstance)

    CDC.ChangeFont "Courier New",13, True, False, False, False

    If ob.Selected Then
        CDC.ChangeLineStyle CFC_SOLID,1,RGB(255,0,0)
    End If

    CDC.Rectangle ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1),ob.CoordinatesY(1)
    CDC.TextOut ob.CoordinatesX(0)+5,ob.CoordinatesY(0)+15, "spreading"

    ob.DrawConnectionPoints

End Sub

Public Function CEM_EMS_Spreading_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Spreading_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Spreading_OnRelationValid=False
        Exit Function
    End If

End Function

Public Sub CEM_EMS_Spreading_OnSimulationInitializeRun(ob As CFCSim_ModelingElementInstance,
RunNum As Integer)
    ob("AmountSpread")=0
End Sub
```

CEM_EMS_Excavator

```
Public Function CEM_EMS_Excavator_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single,
y As Single) As Boolean
    CEM_EMS_Excavator_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Source" Then
        MessagePrompt "Excavator element can only be defined as a child of a source element"

        CEM_EMS_Excavator_OnCreate=False
        Exit Function
    End If

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+80
    ob.CoordinatesY(1)=y+50

    ob.AddAttribute "Quantity", "Number of Excavators", CFC_Numeric, CFC_Single,
CFC_ReadWrite,1,20
    ob.AddAttribute "Productivity", "Ideal Productivity in Cubic Metres per Hour", CFC_Distribution,
CFC_Single, CFC_ReadWrite

    ob("Quantity")=1
    With ob("Productivity").Distribution
        .DistType=CFC_Constant
        .ParameterValue(0)=1000
    End With

    ob.AddConnectionPoint "c1", ob.CoordinatesX(0)-10,ob.CoordinatesY(0)+25,CInput,5
    ob.AddConnectionPoint "c2", ob.CoordinatesX(0)+90,ob.CoordinatesY(0)+25,COutput,5

    ob.AddResource "Excavator",1

End Function

Public Sub CEM_EMS_Excavator_OnDraw(ob As CFCSim_ModelingElementInstance)
    CDC.RenderPicture "EMS_Excavator",ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1)-
ob.CoordinatesX(0),ob.CoordinatesY(1)-ob.CoordinatesY(0)

    If ob.Selected Then
        CDC.Rectangle ob.CoordinatesX(0)-2,ob.CoordinatesY(0)-
2,ob.CoordinatesX(1)+2,ob.CoordinatesY(1)+2
    End If

    ob.DrawConnectionPoints

End Sub

Public Function CEM_EMS_Excavator_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Excavator_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Excavator_OnRelationValid=False
    End If
End Function

Public Sub CEM_EMS_Excavator_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.res("Excavator").NumResources=ob("Quantity")
    ob.AddEvent "Request",True
    ob.AddEvent "Release"
End Sub

Public Sub CEM_EMS_Excavator_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance,
MyEvent As String, Entity As CFCSim_Entity)
    Dim duration As Double

    Select Case MyEvent
    Case "Request"
        If ob.Parent("AmountLoaded")<ob.Parent("AmountToHaul") Then
            If ob.RequestResource("Excavator",entity,1,entity("LPriority")) Then
                'calculate loading duration
                duration= ( entity("Capacity") / ob("Productivity") ) *60

                ob.ScheduleEvent entity,"Release",duration

                Tracer.Trace "Truck entity obtained a  loader: " & entity.ID , "Simulation"
            Else
                Tracer.Trace "Truck entity waiting for a loader: " & entity.ID , "Simulation"
            End If
        End If
    Case "Release"
        ob.ReleaseResource "Excavator", entity
        entity("PayLoad")=entity("Capacity")
        ob.Parent("AmountLoaded")=ob.Parent("AmountLoaded")+entity("Capacity")

        Tracer.Trace "Truck entity completed loading: " & entity.ID , "Simulation"

        ob.TransferOut entity
    End Select
End Sub

Public Sub CEM_EMS_Excavator_OnSimulationTransferIn(ob As CFCSim_ModelingElementInstance,
Entity As CFCSim_Entity, SrcCp As CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)

    If entity("StartTime")>-1 Then
        entity("CycleStat").Collect SimTime-entity("StartTime")
    End If

    entity("StartTime")=SimTime

    ob.OnSimulationTransferIn entity,SrcCp,Dstcp,True
End Sub
```

CEM_EMS_Dump

```
Public Function CEM_EMS_Dump_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As
Single) As Boolean
     CEM_EMS_Dump_OnCreate=True

    If ob.Parent.ElementType <> "CEM_EMS_Placement" Then
        MessagePrompt "Dump element can only be defined as a child of a placement element"

        CEM_EMS_Dump_OnCreate=False
        Exit Function
    End If

    ob.SetNumCoordinates 2
    ob.CoordinatesX(0)=x
    ob.CoordinatesY(0)=y
    ob.CoordinatesX(1)=x+80
    ob.CoordinatesY(1)=y+50

    ob.AddAttribute "Quantity", "Number of Available Locations", CFC_Numeric, CFC_Single,
CFC_ReadWrite,1,20
    ob("Quantity")=1

    ob.AddConnectionPoint "c1", ob.CoordinatesX(0)-10,ob.CoordinatesY(0)+25,CInput,5
    ob.AddConnectionPoint "c2", ob.CoordinatesX(0)+90,ob.CoordinatesY(0)+25,COutput,5

    ob.AddResource "Dump",1

End Function

Public Sub CEM_EMS_Dump_OnDraw(ob As CFCSim_ModelingElementInstance)

    CDC.RenderPicture "EMS_Dump",ob.CoordinatesX(0),ob.CoordinatesY(0),ob.CoordinatesX(1)-
ob.CoordinatesX(0),ob.CoordinatesY(1)-ob.CoordinatesY(0)
```

```
    If ob.Selected Then
        CDC.Rectangle ob.CoordinatesX(0)-2,ob.CoordinatesY(0)-
2,ob.CoordinatesX(1)+2,ob.CoordinatesY(1)+2
    End If

    ob.DrawConnectionPoints

End Sub

Public Function CEM_EMS_Dump_OnRelationValid(srcCP As CFCSim_ConnectionPoint, dstCP As
CFCSim_ConnectionPoint) As Boolean
    CEM_EMS_Dump_OnRelationValid=True

    If srcCP.RelationsTo.Count>0 Then
        MessagePrompt "Only one relation is allowed from this connection point "
        CEM_EMS_Dump_OnRelationValid=False
    End If
End Function

Public Sub CEM_EMS_Dump_OnSimulationInitialize(ob As CFCSim_ModelingElementInstance)
    ob.res("Dump").NumResources=ob("Quantity")
    ob.AddEvent "Request",True
    ob.AddEvent "Release"
End Sub

Public Sub CEM_EMS_Dump_OnSimulationProcessEvent(ob As CFCSim_ModelingElementInstance,
MyEvent As String, Entity As CFCSim_Entity)
    Dim duration As Double

    Select Case MyEvent
    Case "Request"
        If ob.RequestResource("Dump",entity) Then

            ob.ScheduleEvent entity,"Release",entity("DumpingTime").Value

            Tracer.Trace "Truck entity obtained a  dumping location: " & entity.ID , "Simulation"

        Else
            Tracer.Trace "Truck entity waiting for a dumping location: " & entity.ID , "Simulation"
        End If
    Case "Release"
        ob.ReleaseResource "Dump", entity
        ob.Parent("AmountDumped")=ob.Parent("AmountDumped")+entity("Payload")
        entity("PayLoad")=0

        Tracer.Trace "Truck entity completed dumping: " & entity.ID , "Simulation"

        ob.TransferOut entity
    End Select
End Sub
```